

Smart SOA Solutions with WebSphere Enterprise Service Bus Registry Edition V7.5

Discover the advantages of integrating
a service registry with an ESB

Explore how to register services
and implement mediations

Learn by example with
practical scenarios



Martin Keen
Thomas Büttner
Aditya P Dutta
Bernardo Fagalde
Andrew Humphreys
Nay Lin
Fatima Otori
Jesús Ángel Sáenz Viguera



International Technical Support Organization

**Smart SOA Solutions with WebSphere Enterprise
Service Bus Registry Edition V7.5**

August 2011

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (August 2011)

This edition applies to WebSphere Enterprise Service Bus Registry Edition V7.5.

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team who wrote this book	xi
Now you can become a published author, too!	xiv
Comments welcome	xiv
Stay connected to IBM Redbooks	xv
Part 1. Positioning WebSphere Enterprise Service Bus Registry Edition	1
Chapter 1. SOA and the roles of an ESB and service registry	3
1.1 The value of service-oriented architecture	4
1.2 Scenarios that illustrate SOA requirements	4
1.2.1 Adopting SOA to increase flexibility and reduce cost	4
1.2.2 Building a service exposition layer	7
1.3 SOA components	9
1.3.1 Service	9
1.3.2 Mediation	11
1.3.3 Enterprise service bus	13
1.3.4 Integration	14
1.3.5 Service registry and repository	17
1.4 Combining the ESB and service registry and repository	19
Chapter 2. WebSphere ESB Registry Edition solution overview	21
2.1 WebSphere Service Registry and Repository	22
2.1.1 WSRR role in SOA	22
2.1.2 WSRR technical architecture overview	23
2.2 WebSphere ESB	33
2.2.1 WebSphere ESB role in SOA	33
2.2.2 WebSphere ESB technical architecture overview	34
2.2.3 IBM Integration Designer	41
2.3 WebSphere ESB Registry Edition dynamic ESB usage patterns	42
2.3.1 Dynamic service gateway pattern	42
2.3.2 Mediation policy patterns	43
2.3.3 Service level agreement-related patterns	43
Part 2. WebSphere Enterprise Service Bus Registry Edition topologies	45

Chapter 3. Topology overview	47
3.1 Workload concepts	48
3.2 WebSphere ESB Registry Edition Low Workload topology	48
3.2.1 Basic Low Workload topology	49
3.2.2 Advanced Low Workload topology	50
3.3 WebSphere ESB Registry Edition High Workload topology	51
3.3.1 WSRR deployment topologies	51
3.3.2 WebSphere ESB deployment topologies	54
3.3.3 Main High Workload topology for WebSphere ESB Registry Edition	56
3.4 Sizing a WebSphere ESB Registry Edition solution	61
3.4.1 Sizing WebSphere ESB	62
3.4.2 Sizing WSRR	72
3.5 Monitoring the WebSphere ESB Registry Edition architecture	73
3.5.1 IBM Tivoli Composite Application Manager	74
Chapter 4. Implementing a Low Workload topology	77
4.1 Topology description	78
4.2 Topology implementation road map	79
4.3 Checking system requirements	80
4.4 Installing and configuring the runtime components	81
4.4.1 Installing and configuring WSRR	81
4.4.2 Installing and configuring WebSphere ESB	87
4.4.3 Connecting WebSphere ESB and WSRR	97
4.5 Installing and configuring the tools	101
4.5.1 Installing and configuring IBM Integration Designer	102
4.5.2 Installing and configuring the WSRR Eclipse Plug-in	117
4.5.3 Installing and configuring WSRR Studio	124
Part 3. Building WebSphere Enterprise Service Bus Registry Edition solutions	131
Chapter 5. The case study and scenario used in this book	133
5.1 Introduction to the case study	134
5.2 The supply company's SOA vision and requirements	134
5.2.1 Create a standards-based enterprise service bus	137
5.2.2 Enable flexible and dynamic service mediation	138
5.2.3 Ensure governance and policy enforcement	138
5.2.4 Promote service reuse with control	139
5.3 The supply company's SOA governance strategy	139
5.3.1 Foundation phase	141
5.3.2 Pilot phase	141
5.3.3 Limited rollout phase	141
5.3.4 Enterprise-wide rollout phase	142
Chapter 6. Governance enablement profile	143

6.1 Governance enablement profile overview	144
6.2 Models in the governance enablement profile	144
6.2.1 Service model	145
6.2.2 Governance enablement model	150
6.2.3 Governance enablement profile extensions model	159
6.2.4 Manual endpoint model	160
6.2.5 Advanced Lifecycle Edition model	161
6.2.6 Governance enablement model and service model relationships	163
6.3 Roles and access control	164
6.4 Life cycles in the governance enablement profile	166
6.4.1 The asset life cycle	167
6.4.2 The capability life cycle	169
6.4.3 The SOA life cycle	170
6.4.4 The SLD life cycle	178
6.4.5 The SLA life cycle	181
6.4.6 The endpoint life cycle	184
6.5 Policies in the governance enablement profile	185
6.6 Governance profile taxonomy	186
6.7 Plug-in configurations	187
6.7.1 The correlator modifier	188
6.8 Web user interface configuration	191
6.8.1 WSRR web user console	191
6.8.2 WSRR Business Space user interface	192
6.9 Modifying the governance enablement profile	192
Chapter 7. Registering services	195
7.1 WebSphere Service Registry and Repository user interfaces	196
7.2 Using the Business Space interface with WSRR	196
7.2.1 Service Registry Business Space widgets	197
7.2.2 Customizing and configuring the Business Space UI and Service Registry widgets	199
7.2.3 Using the Service Registry for SOA governance template	200
7.3 Preparing Business Space for use with WSRR	207
7.3.1 Creating a new space in Business Space	208
7.3.2 Customizing Business Space to add a Load Documents widget	208
7.4 Registering a service using Business Space	211
7.4.1 Creating an organizational structure	212
7.4.2 Identifying the business capability provided by a service	216
7.4.3 Registering a service	220
7.4.4 Providing scope and planning information for a service	229
7.4.5 Providing a specification and service level definition for a service	232
7.4.6 Governing the service consumer	238
7.4.7 Completing the services life cycle	248

Chapter 8. Implementing a mediation	251
8.1 Addressing the business requirements	252
8.2 Benefits of a mediation proxy module	252
8.2.1 Benefits of a dynamic endpoint lookup in a mediation	255
8.2.2 Benefits of an SLA check in a mediation	256
8.3 Basic primitives to integrate a smart WebSphere ESB Registry Edition solution	256
8.3.1 Endpoint Lookup mediation primitive	257
8.3.2 SLA Check mediation primitive	260
8.3.3 Service Invoke mediation primitive and Callout node	262
8.4 Implementing a basic mediation proxy module	264
8.4.1 Creating the mediation library module	265
8.4.2 Creating the basic mediation proxy module	279
8.4.3 Testing the mediation proxy module	298
Chapter 9. Extending the mediation	315
9.1 Addressing the business requirements	316
9.2 Other benefits of a mediation proxy module	316
9.2.1 Benefits of the SLA Endpoint Lookup primitive	317
9.2.2 Benefits of the Policy Resolution primitive	317
9.2.3 Benefits of the Gateway Endpoint Lookup primitive	318
9.3 Additional primitives for WebSphere ESB Registry Edition	319
9.3.1 SLA Endpoint Lookup mediation primitive	320
9.3.2 Policy Resolution mediation primitive	322
9.3.3 Gateway Endpoint Lookup mediation primitive	326
9.4 Extending the mediation proxy	328
9.4.1 Implementing an SLA-based endpoint resolution	329
9.4.2 Implementing dynamic mediations	336
9.4.3 Testing the solution	346
Chapter 10. Service versioning	353
10.1 Service versioning concepts	354
10.1.1 Service versions	354
10.1.2 WSDL document design and naming standards	356
10.1.3 Service endpoint management	358
10.2 Service versioning management	359
10.2.1 Managing service versions based on SLA	360
10.2.2 Support service versioning with a dynamic SLA gateway solution	364
10.2.3 Publishing the gateway endpoint for each service version	367
10.3 Service versioning scenarios	368
10.3.1 Processing requests for version 1.0 of the service	368
10.3.2 Introducing a new sub-minor service version	377
10.3.3 Introducing a new a minor service version	382

10.3.4 Introducing a new major service version	387
10.4 Summary	391
Part 4. Appendixes	393
Appendix A. Additional material	395
Locating the material	395
Using the material	396
Downloading and extracting the material	396
Related publications	397
IBM Redbooks publications	397
Online resources	397
How to get IBM Redbooks publications	398
Help from IBM	398

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

The following company names appearing in this publication are fictitious:

Supply Company

These names are used for instructional purposes only.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®

DB2®

developerWorks®

Domino®

Global Business Services®


IBM®

iSeries®

Lotus®

Rational®

Redbooks®

Redbooks (logo) ®

Tivoli Enterprise Console®

Tivoli®

WebSphere®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication provides you with a technical overview of IBM WebSphere® Enterprise Service Bus Registry Edition V7.5.

Part 1 outlines the roles of a service registry and an enterprise service bus (ESB), and explains the benefits of combining these technologies.

Part 2 focuses specifically on the ESB and registry that is offered by WebSphere Enterprise Service Bus Registry Edition. It also describes topology choices and installation.

Part 3 presents a fictional business scenario that demonstrates how an organization can register services and build simple and advanced mediations using these services.

IT specialists, IT architects, and those who are looking for a technical discussion of WebSphere Enterprise Service Bus Registry Edition will find value in this book.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

Martin Keen is a Redbooks Project Leader in the Raleigh Center. He writes extensively about WebSphere products and service-oriented architecture (SOA). He also teaches IBM classes worldwide about WebSphere, SOA, and ESB. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, U.K. He holds a Bachelor's degree in Computer Studies from Southampton Institute of Higher Education.

Thomas Büttner is an IT Specialist working for IBM Software Services for WebSphere in Germany. He is based in Cologne and has more than eight years of experience in IT, including four years in Business Process Management and Enterprise Application Integration. He holds a diploma degree in Information Systems Management from the University of Leipzig in Germany. His areas of expertise include WebSphere Application Server, WebSphere Process Server and WebSphere Enterprise Service Bus. Thomas is an IBM Certified Solution Developer for WebSphere Integration Developer, an IBM Certified System

Administrator for WebSphere Application Server, and an IBM Certified SOA Associate.

Aditya P Dutta is an IT Architect at IBM Global Business Services®, India. He provides consulting and architectural assistance to IBM clients. He specializes in BPM, SOA, and Enterprise Integration solutions using IBM middleware products. Aditya holds a Bachelor's degree in Electronics and Instrumentation Engineering from the College of Engineering and Technology, Bhubaneswar, India. He has published and presented at various IBM technical forums including developerWorks® and technical conferences, and has coauthored IBM Redbooks publications.

Bernardo Fagalde is a Senior IT Architect (Integration discipline) working for IBM GBS. He is the SI&E competency leader for Uruguay. He has seven years of experience in the field of SOA and Integration, and has worked for IBM for 10 years. Bernardo holds a Bachelor's degree in Computer Engineering from the public university of Uruguay (Universidad de la República—Facultad de Ingeniería—Ingeniero en Computación). His areas of expertise include SOA, Enterprise Architecture, Integration, J2EE, and the WebSphere family. He has coauthored four IBM Redbooks publications.

Andrew Humphreys is a Consulting IT Specialist based at the IBM Hursley Laboratory in the U.K. He is a Chartered Engineer and holds a Master's degree in Information Systems from The University of Huddersfield and a Bachelor's degree in Economics from City University, London. Andrew has extensive experience in designing SOA- and ESB-style solutions, and in working with clients in a variety of industries using a range of IBM and other technologies.

Nay Lin is a WebSphere IT Specialist at IBM, USA. He has 18 years of experience in software engineering and 10 years of experience in the IT field. He holds a degree in PhD from Arizona State University. Nay's areas of expertise include SOA solutions, SOA governance, and BPM. He has written extensively on these areas.

Fatima Otori is a Mainframe Client Technical Professional based in Chicago, IL. She has five years of experience in the WebSphere field. She holds a Bachelor of Science degree from Tulane University and a Master's degree in Electrical Engineering from the University of Michigan Ann Arbor. Her areas of expertise include IBM WebSphere SOA/BPM Solutions and IBM WebSphere Connectivity Solutions. Fatima has written extensively about Application Modernization, Application Integration, and overall Business Process Improvement and Management.

Jesús Ángel Sáenz Viguera is a WebSphere Technical Specialist at IBM Spain. During his 12 years of experience as an IT professional he has performed various technical roles including J2EE Developer, Technology Consultant and

Product Specialist. Currently Jesús supports technical sales activities for the SOA WebSphere portfolio in Israel, Greece, Portugal and Spain. His areas of expertise include SOA Connectivity, SOA governance and BPM, and Java and XML base technologies. He holds a degree in Technical Engineering in Computer Systems from the Universidad Politécnica de Madrid.

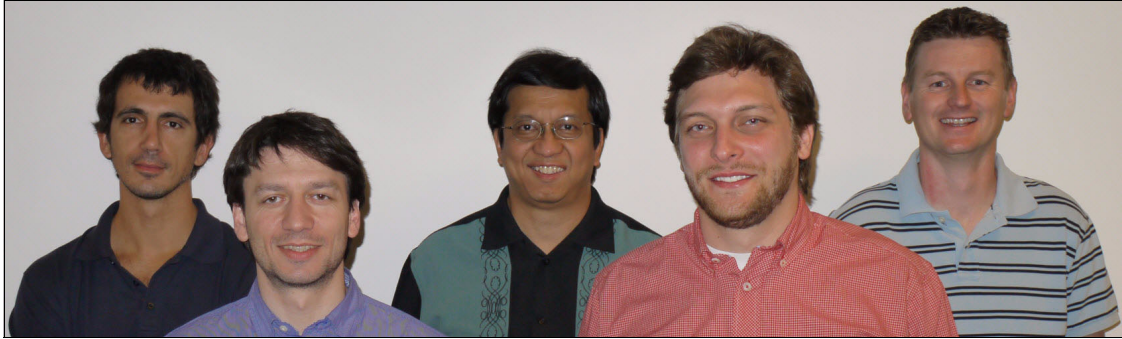


Figure 1 Raleigh team (left to right): Bernardo, Jesús, Nay, Thomas, and Martin

Thanks to the following people for their contributions to this project:

- ▶ Laura Olson, Product Manager, WebSphere Service Registry and Repository
- ▶ Andrew Leonard, IBM WebSphere Service Registry and Repository Development
- ▶ Steve Groeger, IBM WebSphere Service Registry and Repository Development
- ▶ Chris Kalus, IBM WebSphere ESB Development
- ▶ Alex Wood, IBM WebSphere ESB Development
- ▶ Ian Heritage, IBM Worldwide WebSphere Technical Sales Support
- ▶ Phil Norton, IBM WebSphere Messaging Development
- ▶ Gary Chapman, IBM WebSphere Service Registry and Repository Information Development
- ▶ Brian Hulse, IBM WebSphere Service Registry and Repository Development
- ▶ Chris Dudley, IBM WebSphere Service Registry and Repository Development
- ▶ Martin Rowe, IBM WebSphere Service Registry and Repository Development
- ▶ Simon Burns, IBM WebSphere Service Registry and Repository Support
- ▶ Stephen Cocks, IBM WebSphere Connectivity Architect
- ▶ Colin Westlake, IBM Information Development
- ▶ Philip McCormick, IBM WebSphere ESB SVT

- ▶ Joseph Ciervo, IBM Techline
- ▶ Andy Garratt, IBM Software Services for WebSphere
- ▶ Paul Harris, IBM WebSphere ESB Performance
- ▶ Martin Ross, IBM WebSphere ESB Performance
- ▶ Marcelo Marrero, IBM Uruguay
- ▶ David Currie, IBM WebSphere ESB Release Architect
- ▶ Dennis Miller, IBM WebSphere Technical Sales

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>


- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Part 1

Positioning WebSphere Enterprise Service Bus Registry Edition



SOA and the roles of an ESB and service registry

This chapter provides an introduction to service-oriented architecture (SOA). It defines the enterprise service bus (ESB) and service registry as components in an SOA and describes the role that they play in an SOA implementation.

1.1 The value of service-oriented architecture

Organizations spend a considerable amount of time and money trying to achieve rapid and flexible integration of IT systems. This activity usually involves the following objectives:

- ▶ Increase the speed at which businesses can implement new products and processes, change existing ones, or recombine them to deliver new value
- ▶ Reduce implementation and ownership costs of IT systems and the integration between them
- ▶ Simplify the integration work that is required by mergers and acquisitions
- ▶ Achieve better IT use and return on investment
- ▶ Implement business processes at a level that is independent from the applications and platforms that are used to support those processes

Service-oriented architecture (SOA) is an approach that helps to achieve this rapid flexible integration. SOA provides the following value:

- ▶ Reduces IT constraints on the business to increase the company's responsiveness to change
- ▶ Reduces IT costs by increasing reuse and removing redundancy and duplication in IT resources

SOA is based on the concept of a *service*. A service encapsulates a discrete business function and provides a clearly defined interface that makes it easily reusable. Applications are built by invoking services, and services can be chained together to form more complex functions. This flexibility allows companies to quickly implement business processes that meet changing business needs.

1.2 Scenarios that illustrate SOA requirements

This section describes two fictitious companies that are at different stages in SOA adoption. Each company is approaching SOA from a different perspective. We use these scenarios to illustrate the requirements for successful SOA implementation.

1.2.1 Adopting SOA to increase flexibility and reduce cost

The first example is a company that is just starting with SOA. Both the business and the IT teams for the company want to adopt SOA to increase flexibility and to

reduce the cost of developing new applications. The company does not intend to follow a large business process reengineering program to revolutionize its approach to IT and to adopt SOA universally. Instead, they will introduce SOA principles gradually. New applications will be designed and built to invoke services, and new services will be built as required by the new applications.

Evolution of an SOA: SOAs tend to *evolve* rather than to be built. It is rare for a company to run a project specifically to introduce SOA. Instead, SOA implementation tends to happen on a project-by-project basis.

For example, projects are set up to deliver new applications as required by the business. Each project's primary goal is to deliver the new business application, not to implement an SOA. However, in the process of delivering the new applications, the project follows SOA principles in application design by describing each processing step in terms of a service. The project then builds new or reuses existing services to meet that processing requirement.

This process allows the project to deliver the business value of the specific project and to increase the catalog of reusable services that are available to other projects over time. The challenge with this approach is to ensure that the services developed are genuinely reusable for multiple projects and are not tied to the specific requirements of the initial client.

Figure 1-1 illustrates the first SOA-based application that this company plans to introduce. It shows a new business process that is implemented to support clients who place orders from a website.

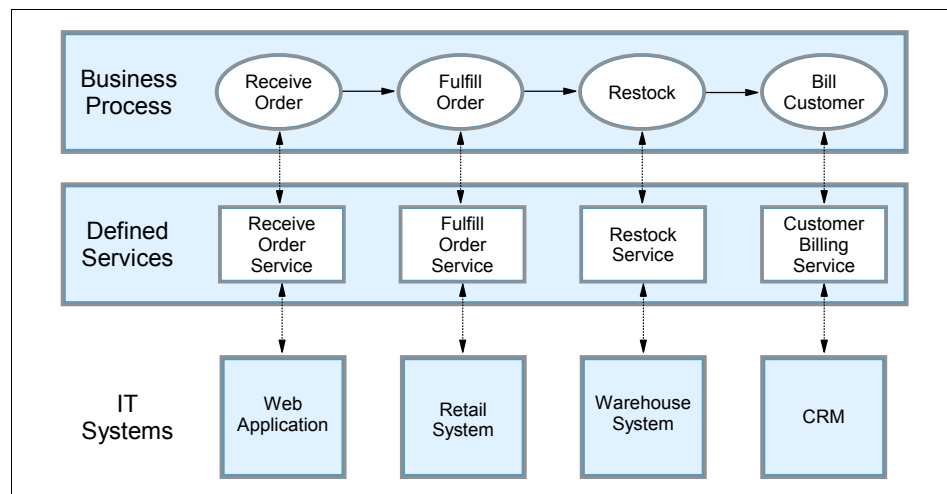


Figure 1-1 A service-oriented approach to building systems

The company already has existing retail, warehouse, and billing systems. It wants to build the new process by reusing the functions that is provided by those systems rather than writing new applications that provide the same functions.

The company defines interfaces to its existing systems in terms of the functions or services that they can offer to support the building of business processes. The defined interfaces simplify building the new front-end web interface to the system. The company then develops an application that makes calls to the services in the correct sequence to complete the new business process.

Following this simple blueprint delivers a service-based application. However, to be successful in SOA, the company realizes that it needs more high-level planning to ensure the required reuse that will deliver true value from its investment in building services.

The company has identified the following needs:

- ▶ The ability to build and host services
 - Expose functions that already exist in back-end systems as services
 - Build new services from scratch
- ▶ An easy method to track the services the company has so it can reuse these services in new projects
- ▶ A mechanism to identify gaps in capability and to commission new services
- ▶ The ability to identify and contact the owners of services to request changes or ask for further information
- ▶ An easy method to understand where services are being consumed so that the company can understand the impact of changing a service on its users
- ▶ The ability to monitor the use of services so that they gather the following information:
 - Identify and troubleshoot unusual patterns of service use
 - Demonstrate successful reuse with a view to increasing investment in SOA
 - Identify services that are operating close to capacity and make business decisions to increase capacity when needed

The company has approached software vendors to discuss recommended software products that can be used to build the SOA infrastructure that can support its needs. Generally, vendors have recommended integration products that can build and host the services. However, the company feels that integration products only partially meets its needs because the products do not provide the services management and monitoring capabilities that the company also requires.

To address the need for services management and monitoring capability, vendors have offered registry and system management products in addition to the integration products. However, the company views the requirements for integration, services management, and monitoring to be intermingled and finds it difficult to separate these components. The company prefers a single, combined solution.

1.2.2 Building a service exposition layer

In this example, a company wants to build a service exposition layer that exposes and manages a consistent set of services for use by business process management (BPM) and SOA applications.

The company is not new to SOA principles. It has previously developed multiple applications that use web services for communication and integration. It has more than 200 web services, but these web services were all developed piecemeal to meet the requirements of individual projects. As a result, the services are mainly point-to-point. The services are designed to solve a specific integration requirement and are not designed for reuse by multiple consumers. Thus, the services do not apply security, logging, and other standards consistently. In addition, there is no central control over who runs or uses the services.

Challenges of adopting SOA piecemeal: Companies that adopted SOA piecemeal with no central control or strategy usually have a large number of point-to-point services with limited reuse. These companies then face the following challenges in getting value from SOA:

- ▶ Critical outages when trying to change services because the service providers do not identify the consumers and cannot determine the impact of changes
- ▶ Performance and availability issues for applications because there is no definition of how the services will perform
- ▶ Duplication of services that are produced for different projects because there is no ability to discover what services exist or any mechanism to review all new service requests for all projects
- ▶ Complexity for SOA application builders because each service that they access uses different data dictionaries, security policies, and so forth

This company is launching a large BPM initiative. It consists of several projects running concurrently that are trying to define, improve, and automate the business processes that they follow. To address the current issues with managing their web services and to support the BPM initiative going forward, it

has set up a central SOA project team that manages and provides the SOA services that are used throughout the company.

The SOA team plans to introduce a service exposition layer from which all services can be accessed and managed. The service exposition layer includes the following functions:

- ▶ Provides the single point where all services are accessed by BPM and SOA applications
- ▶ Introduces standardized interfaces with common data model, naming standards, and qualities of service for all services
- ▶ Introduces SOA-wide policies, for example to enforce that a consumer has entitlement to use the service, which is currently performed ad hoc, depending on how a service is implemented
- ▶ Allows service consumers and providers to create service contracts and ensures compliance with service level agreements (SLAs)
- ▶ Provides management and control functions, such as providing audit and tracking capabilities and monitoring of service performance and availability

The SOA team wants a software solution that they can use to build the software exposition layer. They have identified the following critical requirements:

- ▶ Functional characteristics
 - Ability to configure and enforce policies for security and SLAs
 - Ability to expose existing functionality, for example a stored procedure in a database, an IBM CICS® transaction, an MQ application, SAP BAPI, and so forth as a service
 - Ability to perform message data transformation, for example with XSL Transformation (XSLT)
- ▶ Governance
 - A service catalog
 - Ability to define SLAs between service providers and consumers
 - Ability to notify service consumers of change
 - Service life cycle management to cover defining, creating, deploying, and retiring services
 - Ability to configure thresholds and notifications to track SLA violations

The company concluded that it requires both integration and service management tools to meet all the requirements that it has for service exposition in its SOA.

1.3 SOA components

This section defines SOA components and explains the role they play in an SOA.

1.3.1 Service

A *service* is commonly defined as any function that carries out a business task that can be offered to an external consumer. The consumer invokes the service to use the business function that it provides. Figure 1-2 shows a simplified view of the interaction between a service consumer and provider.

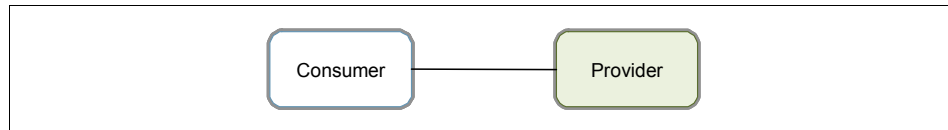


Figure 1-2 Simple service invocation

In addition to implementing a business function, a service in an SOA must also meet the following requirements:

- ▶ Provide a function that is aligned to a business need
- ▶ Be defined by explicit, implementation-independent interface
- ▶ Be invoked through communication protocols that stress location transparency and interoperability

Business aligned function

SOA provides an advantage over earlier approaches to integration because it focuses on business rather than technical requirements. In an SOA, services must offer functions that are aligned to business needs. Simply exposing IT functions by defining the functional interface using, for example, a web service does not make that function an SOA service if the function does not align to a business task. A company might have developed multiple web services; however, if they were developed ad hoc to meet point-to-point integration requirements, they probably do not offer a reusable business function and thus are not SOA services.

For example, a web service that is defined simply to solve a technical integration problem, such as to synchronize data between two systems, can be valuable to meeting a specific integration requirement for the two systems, but it is not a good example of an SOA service. In this case, the service will be too tightly coupled to the specific requirements and details of the interaction between the two systems rather than offering a function that meets the business-level need

and can be used and reused flexibly by multiple business processes rather than specific systems.

Ensuring business alignment of the services requires a focus on *service governance* throughout the SOA life cycle. In an SOA, every service is part of a well-defined *service model* that supports the business processes for the organization, that aligns with a clearly defined business need, that is reusable, that meets quality of service, security, monitoring, logging, and auditing requirements defined at an SOA-wide level, that has an agreed versioning strategy, and that has proper ownership.

Implementation-independent service specifications

When planning for reuse and flexibility with service, using explicit service specifications to define and encapsulate service functions is important. This section describes these service specifications.

Service specification

The *service specification* describes the set of characteristics that the consumer needs to know about the service. It defines the following specifications:

- ▶ The business task performed
- ▶ The technical details of how the consumer interacts with the service in terms of communication protocol, message formats, security and so on
- ▶ Operational aspects, such as response time, throughput, and availability, and various qualities of service associated with interaction

Figure 1-3 shows a modified view of the interaction between a consumer and a provider. Note that the role of the specification is to enable the consumer to understand what the service does and how to invoke the service.

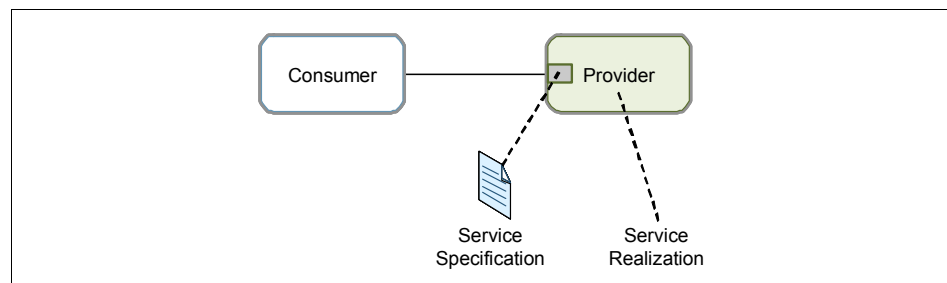


Figure 1-3 Service invocation using service specification

By explicitly defining the interaction in this manner, those aspects of either system that are not part of the interaction, for example the platform on which each is based, can change without affecting the other system. This flexibility

allows either system to change the implementation independently from the service specification, without changing the contract between the provider and consumer.

Service realization

A *service realization* is a physical implementation of a service specification that is physically invoked. The service realization must satisfy all of the characteristics in the service specification in terms of performing the business task and meeting the operational characteristics that are offered in the service specification.

Location-transparent communication protocols

SOA does not specify that a specific protocol is used to provide access to a service. A variety of choices for communication protocols can be used to access services, such as HTTP, HTTPS, JMS, CORBA, and SMTP. Typically, even within a single company, a variety of techniques, products, and protocols are used to address different requirements. A key principle in SOA is that a service is not defined by the communication protocol that it uses but instead is *protocol-independent* so that different protocols can be used to access the same service.

1.3.2 Mediation

Figure 1-1 on page 5 shows services, or more accurately service realizations, directly mapped to functions offered by IT systems that are to be exposed as services. This direct mapping implies that the IT system, as the provider, can satisfy all characteristics that are described in the service specification and can conform with the SOA-wide standards that are defined for a service. In reality, this situation is unlikely to occur. Existing IT systems generally cannot provide functions exactly as described in the service specification and that conform with SOA-wide standards because the systems were not originally designed to provide these functions or conform with the SOA-wide standards.

A *mediation* acts as an intermediary between the provider and the consumer. It provides a mechanism that enable consumers to interact with a provider that has different characteristics from those described in the service specification.

Note: The mediation does not alter the business function that the provider offers. Altering the business functions makes the mediation a provider in its own right, which can lead to issues such as confusion as to where to find the implementation of the business task when it needs to be changed, and who owns it from a change control point of view.

However, a mediation can alter other characteristics, such as changing the communication protocol, translating between data formats, adding encryption and security, making routing decisions, and so on. These characteristics allow the provider's function to be exposed consistently to the consumer with the SOA-wide policies for quality of service, security, monitoring, logging, auditing requirements, and other defined services.

Figure 1-4 shows a modified view of the interaction between a consumer and a provider using a mediation. Note that with a mediation the provider itself is no longer a service realization. The service realization is now a combination of the mediation and the provider.

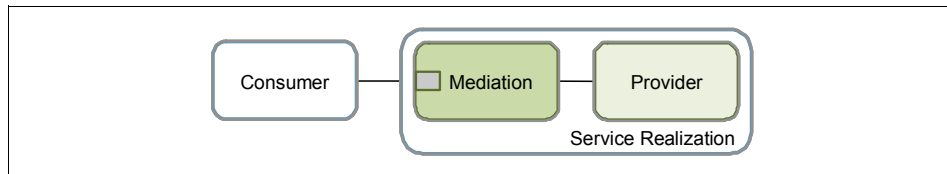


Figure 1-4 Service mediation

Mediations provide a layer of abstraction between the providers of business functions and the consumers of those functions. This layer ensures that the functions are exposed according to the service specifications. Mediations decouple the consumer's view of a service from the actual implementation, increasing the flexibility of the architecture.

For example, a mediation allows the substitution of one provider for another without the consumer being aware of the change or without the need to alter the architecture to support the substitution. The consumer binds to the mediation to access the service, and the mediation maps the request to the location of the real service implementation, with no direct coupling between the consumer and the actual provider of the service.

A mediation is responsible for the following typical tasks:

- ▶ Matching and routing communications between consumers and providers
- ▶ Applying security to service interactions

- ▶ Handling the invocation of alternative service providers in the event that the default provider is unavailable
- ▶ Transforming between different data formats
- ▶ Converting between different transport protocols
- ▶ Logging service invocations for audit purposes

1.3.3 Enterprise service bus

The enterprise service bus (ESB) is commonly defined as the component in an SOA that exposes services to consumers. However, as discussed in 1.3.2, “Mediation” on page 11, exposing provider functions consistently with service specifications is actually the role of a mediation. The ESB is more correctly defined as a *container* for mediations.

Figure 1-5 shows the ESB as a container of mediations that provides a *service exposition layer* of mediations that exposes services to consumers.

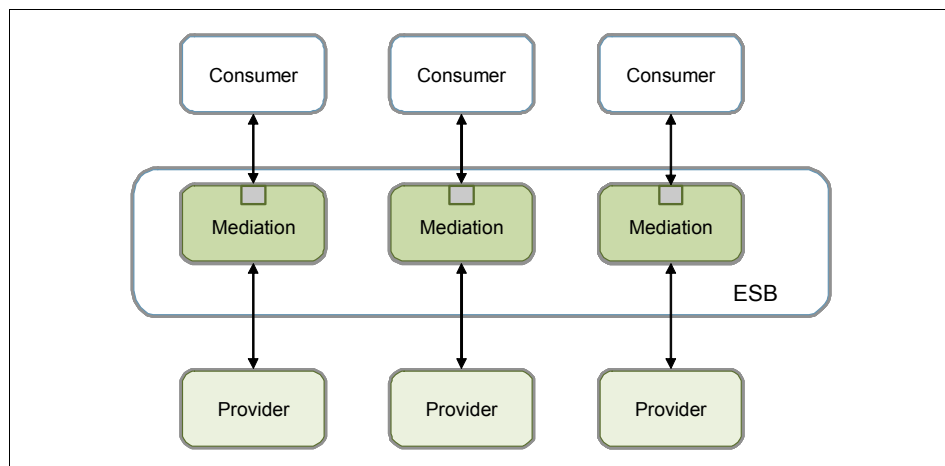


Figure 1-5 ESB service exposition layer

The tasks performed by mediations, such as logging or data mapping, tend to be generic and applied consistently across all mediations in an SOA. This consistency implies that using a middleware product to build and host mediations separately from the systems that are used to host providers simplifies the work that is involved in building the mediations. A middleware product provides a toolkit for coding each mediation consistently rather than coding the same mediation logic separately in each provider system.

ESB products, such as WebSphere ESB Registry Edition, provide development tooling, runtime environments, and monitoring support that simplifies building, deploying, and managing mediations. This simplification allows economies of scale to be realized in the development and management of mediations and makes it easier to implement and run an ESB.

1.3.4 Integration

In 1.3.3, “Enterprise service bus” on page 13, we defined the ESB as a component that hosts mediations. By definition, mediations do not change the business function of the service as offered by the provider. Thus, architecturally at least, the ESB does not host any service providers and does not host any of the logic that is required to implement a business function.

However, the term ESB is commonly used to describe a middleware component that provides a container both for mediations, as with the pure ESB definition, and that also provides integration capabilities. These integration capabilities support the building of new business functions from existing IT systems where the required business function cannot map directly to an existing capability in an IT system. In this case, additional business logic is required to provide the business function. In fact, the majority of software products that are used to implement ESBs provide extensive support for building this logic and for solving integration challenges that are strictly not part of the SOA.

The requirements for integration arise because the business functions that we want to expose as SOA services tend to be embedded in existing IT systems and cannot be exposed easily in a way that meets all the characteristics that are required for the function to be exposed as an SOA service.

An *integration layer* is required in SOA to contain the integration logic needed to expose the business function as an SOA service. ESB products, such as WebSphere ESB Registry Edition, have many capabilities that are aimed at simplifying integration tasks. Using WebSphere ESB Registry Edition to implement an integration layer is a best use case for the product. However, it is important to maintain the separation of concerns between service exposition and integration.

Figure 1-6 details an architecture that shows this separation with the following four layers:

- ▶ Consumer layer
- ▶ Service exposition layer
- ▶ Integration layer
- ▶ Provider layer

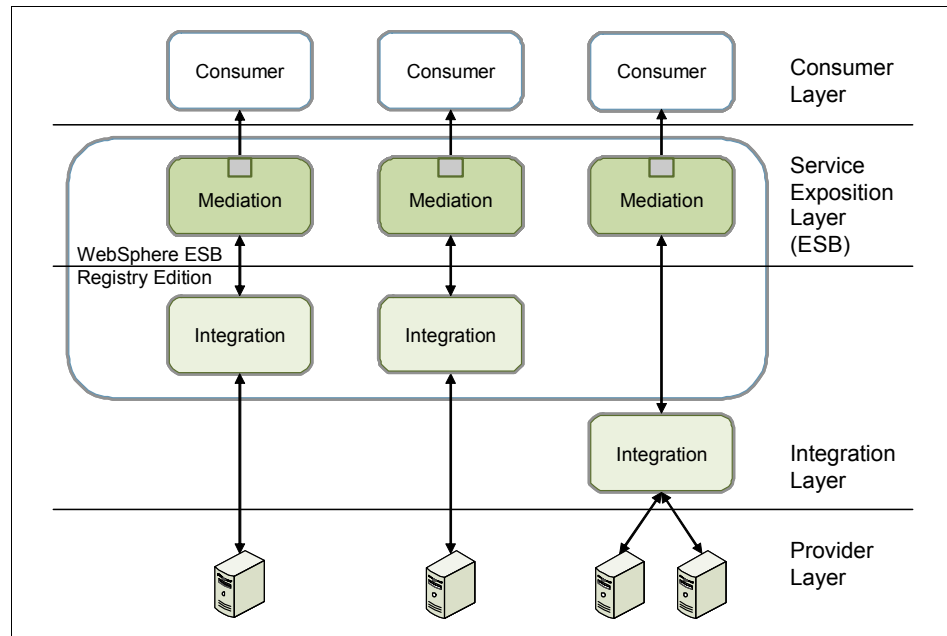


Figure 1-6 ESB responsibilities

The *consumer layer* consists of applications and processes that are constructed from the services that are exposed in the service exposition layer.

The *service exposition layer* consists of mediations, which ensure that providers that are exposed to consumers meet the policies that are defined within the SOA.

The *integration layer* provides the integration that is required to implement a service provider from one or more IT functions. The integration layer can also be used to meet other integration requirements in the enterprise that are not part of the SOA, for example web services that synchronize data that is held in two systems.

The *provider layer* is composed of application software systems. The systems provide fine-grained IT interfaces to allow access to data, but these interfaces are not service based. They can be defined as web services, or they can use other

protocols. They are not encapsulated as true business function that meets the requirements of SOA services.

You can use WebSphere ESB Registry Edition to implement two of the layers:

- ▶ In the service exposition layer, WebSphere ESB Registry Edition hosts and manages all mediations that are required to expose providers to consumers.
- ▶ In the integration layer, WebSphere ESB Registry Edition hosts and manages the integration logic that is required to build providers from functions that are implemented in systems in the system layer.

Integration logic note: Figure 1-6 shows integration logic that is not hosted by WebSphere ESB Registry Edition. Not all integration requires the functions that are offered by ESB products. For example, choreography of multiple back-end system calls might be achieved more efficiently by process choreography tools that are designed specifically for this requirement.

As discussed in 1.3.3, “Enterprise service bus” on page 13, it is important in SOA to separate integration logic and mediations into separate layers. Mediations can be used to implement functions that do not change the business task that is offered by the provider, but which are required to ensure that the service is exposed in accordance with the governance policies in the SOA. For example, consider the following mediation responsibilities:

- ▶ Logging service invocations
- ▶ Determining the endpoint to invoke for a consumer request
- ▶ Managing retries of failed provider invocations
- ▶ Enforcing security on the service as exposed to the consumer
- ▶ Monitoring consumer service level agreements (SLAs)
- ▶ Mapping and transforming data between the service interface that is exposed to the consumer and the interface that is exposed to the provider, assuming that doing so does not change the semantics of the service
- ▶ Setting security to allow a provider to be called correctly, which might involve passing the security credentials from the consumer or mapping between security domains.
- ▶ Switching communication protocols, for example exposing a database, a CICS transaction, or an MQ application as a web service
- ▶ Caching results that are received from a provider call for performance or cost reasons

Integration is about changing the business function that is offered, which creates a new provider rather than just mediating between the consumer and the provider. Consider the following examples of integration logic:

- ▶ Making a service interaction idempotent
- ▶ Augmenting a message with the data that is retrieved (for example from a database) before calling a service
- ▶ Carrying out data mapping and transformation between the service interface that is exposed to the consumer and the interface to the back-end system that changes the semantics of the interaction
- ▶ Adding mediation policies to allow for dynamic processing in a service
- ▶ Exposing new functionality by calling multiple other services, in sequence or parallel

Using ESB software products: Use ESB software products only for simple, mainly synchronous service composition. Use process choreography tools to implement more complex and mainly asynchronous composition.

1.3.5 Service registry and repository

The *service registry and repository* is a key component for gaining visibility of and control over an SOA implementation. It provides a container to store service specifications and provides a catalog of services and service metadata. Consumers can search the registry to find services that they want to use and can download the service specification from the repository to understand how to invoke the service. The registry is used to establish a catalog of existing services, providing visibility into capability and reducing redundancy.

Figure 1-7 shows how a consumer can use a registry and repository to search for and retrieve a service specification and use it to understand how to invoke a service offered by a provider.

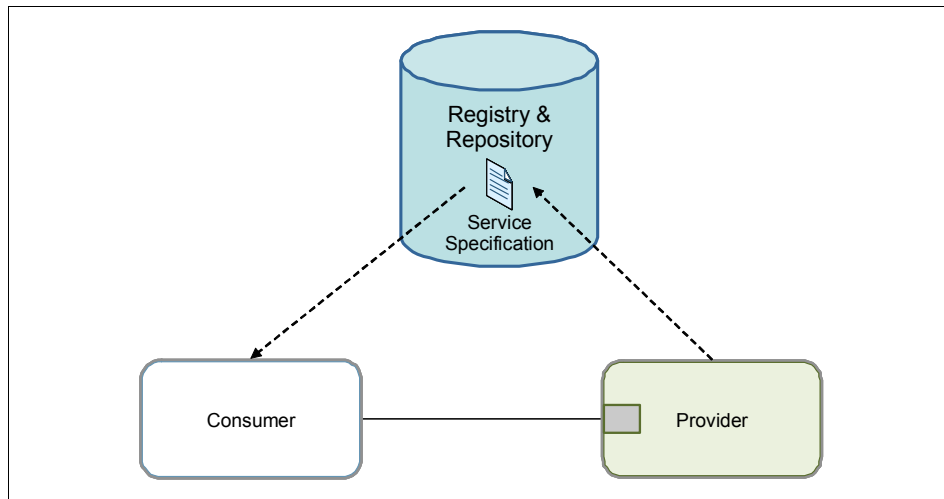


Figure 1-7 Service registry

The basic requirement of a registry and repository is to provide a simple storage location for service specifications. However, the technology used to implement them, such as the registry and repository component in WebSphere ESB Registry Edition, is often more sophisticated than just containing metadata about services. It provides features to help analyze relationships between consumers and providers and provides support for governance around service adoption and versioning and impact analysis.

The evolving nature of an SOA: An SOA is not static. You do not build it and you are done. Services evolve, consumer usage changes, you need to develop new services and retire other services, and you need to track consumers of services. As new service requirements are identified, you need a place where these service requirements are captured and assessed. Ideally, a service registry and repository also supports fully governed service life cycle management, including the service specification and associated metadata, to control the development and management of services in the SOA. This management helps to ensure standardization in services and prevents duplicate services from being developed and deployed.

The primary features the registry and repository should support are listed here:

- ▶ Service discovery
 - Provides a catalog of which services exist and are available for use
 - Shows the life cycle state that a service is in, for example: implemented, in testing, deprecated, and other states
- ▶ Subscription management
 - Shows who is using which services
 - Provides mechanisms to see which services are being reused and which are not being used

These features allow you to use the registry to perform these tasks:

- ▶ Understand the impact of changes to existing services
- ▶ Keep service consumers informed of changes (a service consumer being an application, another service, a business process, and so forth).
- ▶ Obtain real-time visibility of services performance and availability
- ▶ Measure and demonstrate return on investment by tracking service use
- ▶ Provide a mechanism to formalize contracts (SLAs) between service providers and consuming applications

1.4 Combining the ESB and service registry and repository

In 1.3.3, “Enterprise service bus” on page 13 and 1.3.5, “Service registry and repository” on page 17, we discuss why the ESB and registry are both important infrastructure components in an SOA. However, deploying an ESB and service registry as separate components does not fully deliver the promise of SOA. The ESB and service registry must work together to deliver a smarter SOA infrastructure.

Developers often find that it is a challenge to change mediation code quickly in response to changing business requirements. Redevelopment and testing effort is required. It is also complex to synchronize the mediations with the runtime status of the services to invoke. ESB development inherently involves invoking numerous endpoints. When endpoints are not decoupled from mediation code, promotion to other development environment requires significant changes to the ESB configuration.

Enabling dynamic and efficient access to services, policies, and metadata information allows mediations to choose service providers and endpoints dynamically and to adapt quickly to the status of critical services, thereby greatly increasing the flexibility of the ESB.

Storing and defining various application and translation logic outside of the ESB means that it can be updated without changing and redeploying the mediations. Similarly, storing and managing policies outside of the ESB means that the policies can be defined and managed separately from the mediations. The ESB can then query the service registry at run time to determine the policies that are required for the service and can then enforce them.

To address these issues, the ESB and service registry need to interact at run time to make the mediations more dynamic and adaptable. This interaction allows changes to configuration data that is stored in the registry to change the behavior of mediations without updating and redeploying mediation code.

When an ESB and registry work together:

- ▶ Dynamic endpoint selection becomes possible, improving operational flexibility
- ▶ New services that are delivered by the ESB can be published to the registry automatically for lookup and reuse by more consumers
- ▶ Changes to services can be governed

In WebSphere ESB Registry Edition, the ESB and service registry components are integrated through the EndpointLookup, SLACheck, SLAEndpointLookup, Policy Resolution, and Gateway Endpoint Lookup mediation primitives. These primitives are key to accelerating robust flexible mediations that can be changed rapidly without redeploying the mediation, helping to deliver a smarter, more flexible SOA.



WebSphere ESB Registry Edition solution overview

WebSphere ESB Registry Edition is a packaged offering of WebSphere Service Registry and Repository and WebSphere ESB. This chapter introduces this edition.

2.1 WebSphere Service Registry and Repository

WebSphere Service Registry and Repository (WSRR) is a key enabler for SOA Governance. The following section provides an overview of WSRR.

2.1.1 WSRR role in SOA

WSRR is a master metadata repository for service descriptions. It uses a broad definition of services:

- ▶ Traditional web services that implement WSDL interfaces with SOAP or HTTP bindings
- ▶ A broad range of SOA services that can be described using WSDL, XSD, and WS-Policy decorations but that might use a range of protocols and be implemented according to a variety of programming models

As the integration point for service metadata, WSRR establishes a central point for finding and managing service metadata that is acquired from a number of sources, including service application deployments and other service metadata and endpoint registries and repositories, such as UDDI. WSRR brings together service metadata that is scattered throughout an enterprise to provide a single, comprehensive description of a service.

When there is a single description of service, visibility is controlled, versions are managed, proposed changes are analyzed and communicated, usage is monitored, and other parts of the SOA foundation can access service metadata with the confidence that they have found the copy of record. In this context, WSRR handles the metadata management aspects of operational services and provides the system of record for these metadata artifacts.

WSRR provides *registry* functions that support publication of metadata about the function. The requirements and semantics of services allows service consumers to find services or to analyze their relationships. In addition, it provides *repository* functions to store, manage and assign a version number to service metadata. It also enables *governance* of service definitions by providing the following functions:

- ▶ Control access to service metadata
- ▶ Model life cycle of service artifacts
- ▶ Manage promotion of services through phases of their life cycle in various deployment environments
- ▶ Perform impact analysis and communicate changes to the governed service metadata

WSRR adds value throughout all phases of the SOA life cycle:

- ▶ Model and Assemble phases

WSRR is used to locate the copies of record of candidate service interaction metadata or intermediaries and policies that govern the interactions. WSRR can also be used to publish and govern service metadata about emerging, to-be-deployed services.

- ▶ Deployment phase

WSRR provides the system of record for metadata that describes service interaction endpoints. It is populated with metadata as part of SOA solution deployment or through the discovery of existing endpoints.

- ▶ Manage phase

Operational management and resilience in the SOA is enhanced by sharing the service metadata that exists in WSRR with operational data stores.

2.1.2 WSRR technical architecture overview

WSRR is a Java Platform, Enterprise Edition application based on the Java platform and associated platform services that are provided by WebSphere Application Server. WSRR uses a relational database as a store for service metadata (for example, IBM DB2® or Oracle).

The registry and repository component provides basic service metadata storage. It also provides update and retrieval functions that support Create, Retrieve, Update, Delete, and Query capability for service metadata that is stored in the database according to the WSRR content model.

Figure 2-1 illustrates the key components of WSRR.

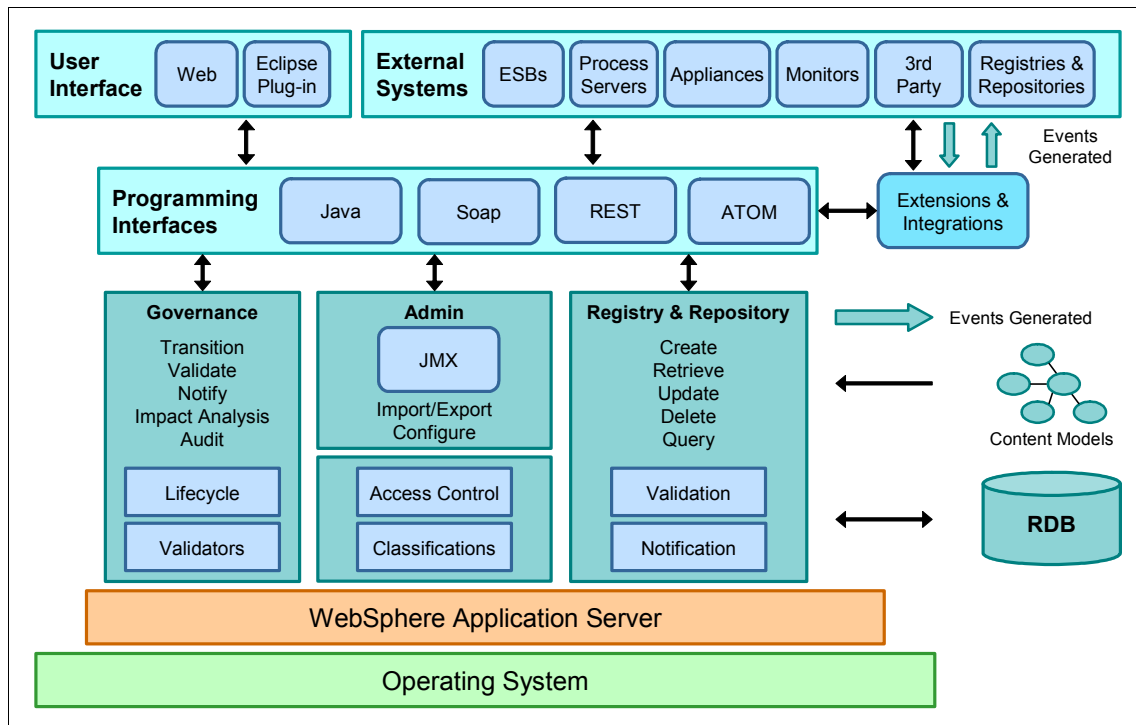


Figure 2-1 WSRR key components

The following sections describe the key components and functions of WSRR.

Registry and repository

WSRR functions as both a registry and a repository.

The repository allows users to store, manage, and query content of documents that hold service metadata descriptions (WSDL, XSD, WS-Policy, SCDL, or XML documents). The repository stores the documents that contain service metadata and also provides a fine-grained representation of the content of those documents (for example, ports, or portTypes in WSDL documents).

The repository also provides registry functions for populating registered service declarations and elements of the derived content models with user-defined properties, relationships, and classifiers. A rich query interface uses these user-defined attributes when users want to find a service endpoint, interface description, or other metadata about a service.

WSRR allows users to plug in validation, notification, and modification functions that are run when changes are made to the repository content (for example, checks for completeness of a service definition). It also provides notifications of any changes to the content of the repository and allows users to register their interest in consuming those notifications.

WSRR includes a default notification handler that publishes change events on a JMS topic. The event specifies the type of event (create, update, delete, or transform), the artifact impacted (identified by its URI), and a few more bits of information about the artifact. To avoid access control issues, the content of the artifact is not shipped with the event but is retrieved separately.

User interfaces

The following user interfaces are provided to access WebSphere Service Registry and Repository:

- ▶ A web user interface
- ▶ An Eclipse plug-in
- ▶ A Business Space user interface

The main interface is a web application that is deployed with the WSRR run time. This servlet-based web user interface is the main way for users representing different roles to interact with WSRR. It supports lookup and publish scenarios, includes metadata management and analysis scenarios, and provides functions that support SOA governance, including import/export and impact analysis.

The web user interface supports customization of the views on the WSRR content that is represented to a user. A set of user interface definition files describes content and layout of the various components that make up the WSRR web interface. The concept of user-role-specific perspectives is supported. WSRR comes with a set of predefined perspectives for the most common user roles, but WSRR users can customize the predefined roles or create role-specific perspectives.

The web user interface also provides administrative functions that allow users to customize the components that control the configuration of the WSRR system, manage access control, and create and modify classification systems.

A subset of this user interface is offered as WSRR Studio, an Eclipse plug-in, to meet the needs of developer and analyst roles that use Eclipse-based tools. The Eclipse plug-in is used primarily for performing lookup, browse, retrieve, and publish tasks. The web-based user interface is used for performing service metadata management and governance.

The Business Space user interface is a browser-based interface that provides widgets that allow you to search, view details of items in the registry, perform key

tasks, and view service reuse data. There are also widgets for viewing data relating to the enforcement of governance validation policies, and WS-I compliance of WSDL documents. Business Space can also be used to modify details of items in the registry. We use Business Space extensively in this book to interact with WSRR.

Programming interfaces

WSRR supports the following types of APIs that can be used to interact with WSRR:

- ▶ Java based
- ▶ SOAP based
- ▶ Representational State Transfer (REST) based
- ▶ Atom based APIs

All APIs support publishing (creating and updating) service metadata artifacts and the metadata that is associated with those artifacts, retrieving service metadata artifacts, deleting the artifacts and their metadata, and querying the content of WSRR.

Basic create, retrieve, update, delete, and governance operations, and a flexible query facility based on XPath, are provided through these APIs. When the SOAP API is used, content is communicated using XML data structures. When the Java API is used, content is communicated using SDO data graphs. When the REST interface is used, content is communicated using XML data structures or JSON data structures.

Clients are provided for the Java and SOAP APIs. Service Data Objects (SDO) capture the data graphs inherent in the content model, allowing access to physical documents, logical parts of the physical documents, and concepts.

The Query API allows the use of XPath expressions to perform unanticipated coarse-grained and fine-grained queries. Queries can be performed using semantic annotations, properties, and all or parts of physical service metadata artifacts. Fragments of metadata can be returned (such as properties for name or endpoint address), all metadata can be returned (data graph), and metadata and documents can be returned. In addition to free-form XPath-based queries, a set of predefined queries are offered to address common paths through the WSRR content model.

The WSRR Atom Web 2.0 APIs allows access to WSRR data from Web 2.0 clients. There are two types of Atom APIs:

- ▶ The Atom Web 2.0 classification API
- ▶ The Atom Web 2.0 configuration API

You can use the WSRR Atom Web 2.0 classification API to retrieve Atom representations of classification systems that are stored in WSRR and the WSRR Atom Web 2.0 configuration API to work with WSRR configuration items.

Administration

The WSRR administration interfaces support import and export of WSRR content for exchange with other metadata repositories (for example, other WSRR installations) and provide a JMX-based API for WSRR configuration and basic administration. WSRR supports a fine-grained access control model that allows users to define which user roles can perform what kind of actions on which artifacts.

WSRR allows users to define and import Classifier systems from basic classifier sets to taxonomies and classification hierarchies. WSRR also provides a JMX-based Administration API that supports basic configuration, and loading and managing metadata in support of WSRR content classification and life cycle management. The API allows users to load definitions of state machines to be used to model the life cycle of governed entities and loading of OWL-described classifier systems.

In addition, the Administration API supports registration of plug-ins for validation and modification functions, and notification providers. Validation functions can be used to control basic create, retrieve, update, and delete operations and in the context of life cycle state transitions for governed entities.

Content model

One of the core functions of WSRR is to manage service descriptions and associated metadata. WSRR has a rich and standards-based content model to support this. All WSRR content elements have a WSRR-assigned URI, a name, and a description.

Figure 2-2 shows an overview of the types of metadata that are stored in WSRR.

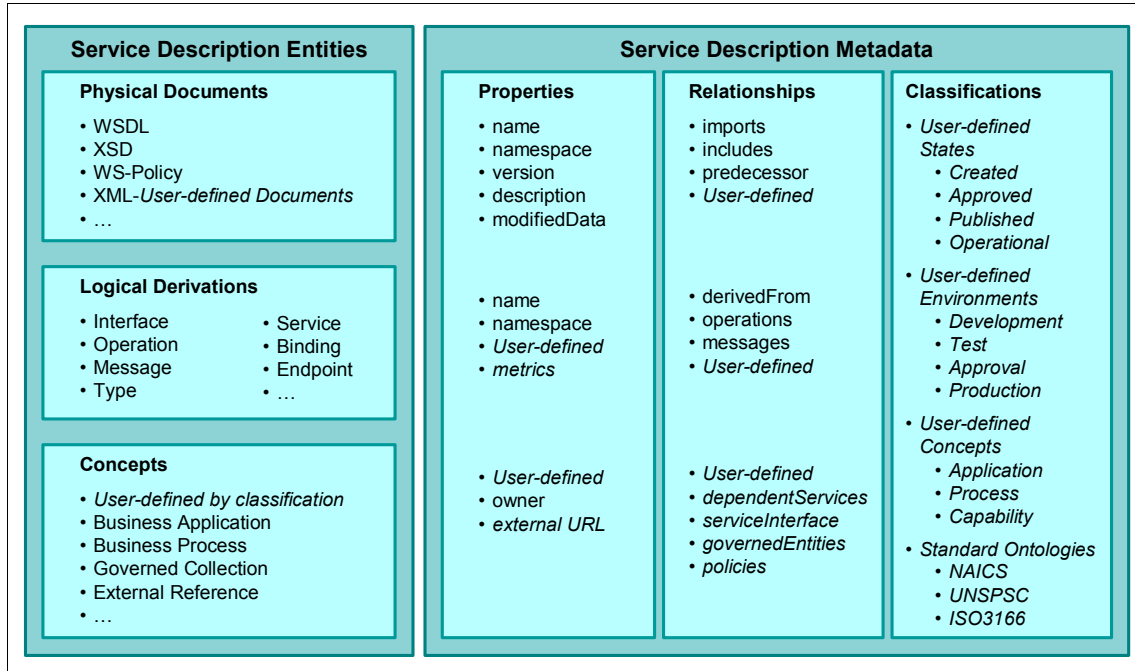


Figure 2-2 WSRR metadata types

As shown in the figure, WSRR uses the following types of metadata:

► **Physical documents**

The most elemental building blocks for the WSRR content model are service metadata artifact documents (physical documents), such as XSD or WSDL files.

► **Logical derivations**

Logical derivations (or logical objects) allow users to explore WSRR content beyond the boundaries of the files stored.

- Concepts

A Concept is the abstract term for the WSRR technical entity called Generic Object. It represents an entity held in WSRR that does not have a physical document associated with it. However, it can be attached with metadata (properties, relationships and classifications) to describe whatever is required.

- Service description metadata

In addition to content directly related to service metadata documents, WSRR supports a number of user-defined metadata types that are used to enhance the service metadata to explain their semantics, which are called service description metadata.

- Classifications

User-defined category systems are imported and shared through the use of documents encoded by using the Web Ontology Language (OWL).

- User-defined properties and relationships

User-defined properties and relationships can be used to customize the set of predefined properties and relationships provided in the WSRR metamodel.

- Business models

Business modelling allows you to represent the objects that are relevant to your organization inside WSRR.

- Templates

The template model allows user-defined properties and relationship names to be applied consistently. WSRR provides a basic template model.

Support for user roles

WSRR provides support for the following user-defined roles so that you can control access to WSRR and its content.

- Java 2 Platform, Enterprise Edition (J2EE) roles

WSRR uses J2EE roles defined in WebSphere Application Server to distinguish between administrative functions and typical service metadata management functions. These roles control access to the programming interface operations.

- Access control roles

WSRR provides a role-based access control that can integrate with the enterprise user directory (through WebSphere Application Server security). The client-defined roles of significance can be defined by policies based on directory groups or other subject properties.

- ▶ User interface perspectives

The WSRR web UI provides the ability to be configured according to user perspectives. A user's role defines the permitted perspectives that are displayed in the web UI, thus allowing the user experience to be personalized according to user's role. Business Spaces and pages can be personalized and access granted based on user role.

- ▶ Default life cycle roles

WSRR comes with a default governance life cycle and access controls to support this life cycle. The roles supported for this are WSRRAdmin and WSRRUser.

Governance

WSRR plays an enabler role for SOA governance, offering increased visibility, with control over reuse, service compliance validation, and change impact analysis.

Governance is supported in the WSRR content model through predefined properties and relationships, state machines, and a number of API and UI functions that can be used in governance processes. Life cycle phase, operational state, and service definition status are all used to offer a mechanism for use, reuse, visibility, and operational readiness.

The governance functions include a life cycle model for governed entities using a state machine that describes these states and valid transitions between them. The governance API also provides validation, modification, and notification plug-ins to guard the transition and the actions to be taken as a result of the transition. WSRR provides interfaces to analyze the impact of changes to WSRR content and the auditing of such changes.

The governance API allows users to perform impact analysis of changes to specific artifacts. A set of predefined impact queries supports patterns of navigation through the WSRR content, such as which WSDL files import or use this XSD. In addition, the governance API provides operations to request life cycle transitions for a governed entity and configuration of email notifications for users who are interested in specific WSRR content changes.

Figure 2-3 illustrates an overview of the governance features.

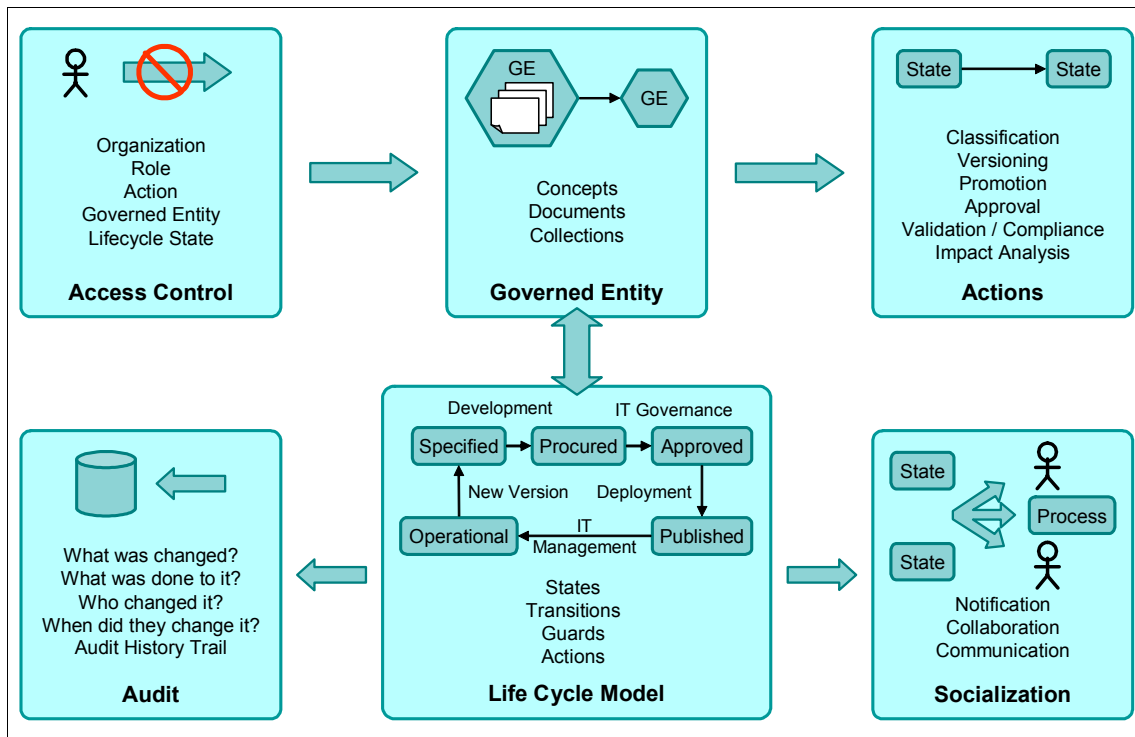


Figure 2-3 WSRR governance features

WSRR includes the following governance features:

- **Governed entity**

The concept of a *governed entity* is at the center of the WSRR governance model. Concepts, documents, and collections can be governed entities. Users can decide to govern a WSDL document or a collection of documents, (for example a WSDL document with related XSD documents, WS-Policy documents, and XML documents that are associated with the WSDL document through user-defined relationships).

- **Life cycle model**

WSRR provides a set of example *state machine definitions*. For each governed entity, a state machine is defined that describes the life cycle states that the entity can take, the transitions between those states, conditions for the transitions to be taken, and actions to be taken when a transition is performed. WSRR provides a set of example state machine definitions, and asset managers can customize these definitions or define new ones as needed to model the life cycle of governed entities.

The WSRR governance API allows users to request that a transition is performed on a governed entity. If the entity is not in a state that allows the transition, the request is rejected. Otherwise, the conditions for the transitions are verified. These conditions can include access control restrictions that prohibit the requester from performing the transition, or the conditions can specify that validators are run before the transition is performed (for example, by checking that certain documents are present in a governed entity collection). If the verifications are successful, the transition is performed and associated actions are run. These actions can include creating a notification event that reports on the transition, or updating to the change of the visibility of the governed entity.

Note: The life cycle states that are modeled in the state machine are WSRR classifiers and can be used in WSRR queries.

► Communication and audit

The WSRR governance component builds on the notification facilities that are provided by the registry and repository allowing life cycle transitions to be notified to users, which provides a high level event of significance to the SOA governance processes.

► Governance API

The WSRR API exposes governance functions, such as impact analysis queries and explicit requests, for running content validators. WSRR provides the following functions in support of governance activities:

- The ability to provide OWL-based user-defined semantic classifiers on all parts of the content model, including operations, data types, and interfaces, to provide consistent and agreed upon terminology throughout the enterprise and to facilitate the sharing of service metadata
- Control of visibility over and access to service metadata for sharing and reuse
- Support for the tracking of service metadata as it makes its way through its governed life cycle, including approvals, deprecation, and retirement, in development, test, staging, and production environments
- The ability to perform automatic and upon-request user-defined validation of state transitions
- A dependency analysis function to assist in the assessment of the impact of a change, capturing many dependencies automatically, as a side effect of storing artifacts in WSRR

- Change notifications using user-defined notification schemes, and basic JMS publication of events providing information about what happened, and an email-based notification feature

2.2 WebSphere ESB

WebSphere ESB is a key enabler for SOA in an enterprise. This section provides an overview of its role in SOA and its technical architecture.

2.2.1 WebSphere ESB role in SOA

WebSphere ESB provides the capabilities of a standards-based enterprise service bus. It is an SOA integration platform that is built on a uniform invocation programming model and a uniform data representation model.

WebSphere ESB manages the flow of messages between service requesters and service providers. Mediation modules within the ESB handle mismatches between requesters and providers, including protocol or interaction-style, interface and quality of service mismatches. In an SCA-based solution, mediation modules are a type of SCA module. The mediation modules perform a special role, and therefore have slightly different characteristics from other components that operate at the business level.

Mediation components operate on messages exchanged between service endpoints. In contrast with regular business application components, they are concerned with the flow of the messages through the infrastructure and not just with the business content of the messages. Rather than performing business functions, they perform routing, transformation, and logging operations on the messages. The information that governs their behavior is often held in headers flowing with the business messages. The IBM SOA programming model introduces the service message object (SMO) data structure to support this pattern.

WebSphere ESB supports advanced interactions between service endpoints on three levels: broad connectivity, a spectrum of interaction models and qualities of interaction, and mediation capabilities. The product supports connectivity between endpoints through a variety of protocols and APIs:

- ▶ Java Message Service (JMS) 1.1 applications can exploit a variety of transports, including TCP/IP, SSL, HTTP, and HTTPS
- ▶ WebSphere ESB supports the following web services standards that enable applications to make use of web service capabilities:

- SOAP/HTTP, SOAP/JMS, WSDL 1.1
- UDDI 3.0 Service Registry
- WS-* Standards including WS-Security, WS-Atomic Transactions

Because it is built on WebSphere Application Server, WebSphere ESB can provide smooth interoperability with other products in the WebSphere portfolio, including WebSphere MQ and WebSphere Message Broker. It can also, with WebSphere Adapters solutions, use existing application assets and capture and disseminate business events.

The message clients for C/C++ and Microsoft .NET enable non-Java applications to connect to WebSphere ESB using an API similar to the JMS API.

Other features at the connectivity level perform basic protocol conversion between endpoints where the protocol used by the requester to dispatch requests (such as SOAP over HTTP) is different from that of the service provider that is to handle those requests (such as SOAP over JMS).

2.2.2 WebSphere ESB technical architecture overview

The base runtime infrastructure for WebSphere ESB is WebSphere Application Server. The Service Component Architecture (SCA) and business objects that are part of the IBM SOA core provide the uniform invocation and data-representation programming models. The SOA core includes the Common Event Infrastructure for generating events for the monitoring and management of applications running on WebSphere ESB.

The combination of a powerful foundation (WebSphere Application Server and the SOA Core) and service components in WebSphere ESB allows quick development and deployment of sophisticated composite applications that run on WebSphere ESB.

In Figure 2-4, the Mediation Flows, Maps, and Relationships provide the essential functions for WebSphere ESB. SCA, Business Objects, and CEI are available as the base for the WebSphere ESB platform.

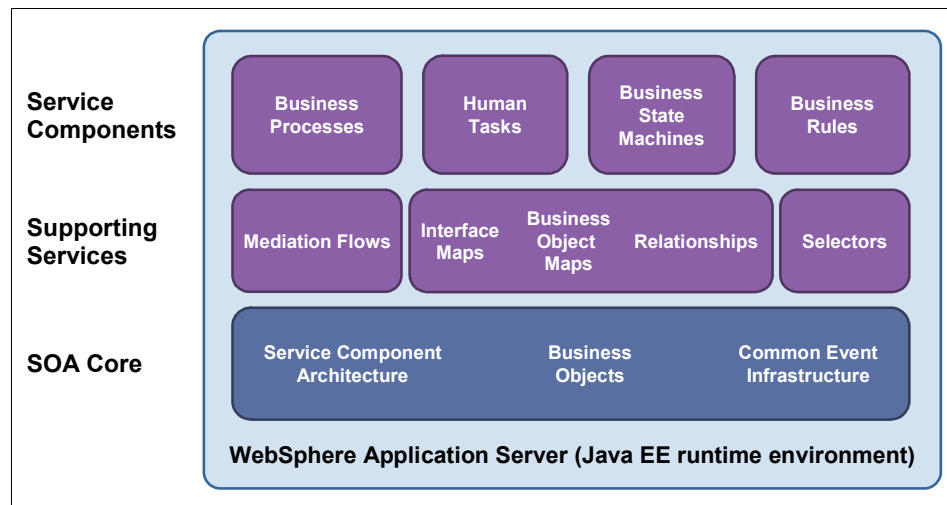


Figure 2-4 Key components of IBM SOA services

Business Processes, Human Tasks, Business State Machines, Business Rules, and Selectors are not part of WebSphere ESB. They are part of WebSphere Process Server, which is built on top of all SOA Core and supporting services.

SCA and service components

An SCA is an architecture in which all elements of a business transaction, such as access to web services, enterprise information system (EIS) service assets, business rules, workflows, databases and so on, are represented in a service-oriented way. In SCA, a service component, also called an *SCA component*, defines a service implementation. All integration artifacts running on WebSphere ESB are represented as components with well-defined interfaces. Each service component has an interface, and you can wire service components together to form a module that is deployed to WebSphere ESB.

This setup creates a flexible runtime environment and enables changing any part of an application without affecting the other parts. For example, by replacing the service components in the assembly diagram, you can replace a human task that represents an approval with a business rule that represents an automatic approval.

Service components interact with existing applications using the following programming constructs:

- ▶ Java Beans
- ▶ Enterprise Java Beans
- ▶ Web Services
- ▶ JMS Messages

In addition, service components can interact with other applications on enterprise information systems (EIS) with WebSphere Adapters.

Business objects

Business objects define the data that flows between components defined in the SCA. As part of the WebSphere Application Server capabilities that are built in to WebSphere ESB, business objects provide a framework for data application development that simplifies the Java EE data programming model. The business object framework, included in WebSphere ESB as part of the SOA core, provides a universal means of describing and exchanging data between SCA services (for example, JDBC ResultSet and XML Schema described data).

A *business object* is a set of attributes that represent a business entity (such as Employee), an action on the data (such as a create or update operation), and instructions for processing the data. Components of the integration application use business objects to exchange information and trigger actions. Business objects are flexible because they can represent any kind of data.

For example, in addition to supporting the data canonizations model of traditional integration servers, they also can represent data that is returned from a synchronous EJB Session Bean facade or a synchronous business process, and then they can be bound to WebSphere Portal portlets and JavaServer Faces (JSF) components.

Business objects are the primary mechanism for representing business entities or for documenting literal message definitions. Business objects enable everything from a simple basic object with scalar properties to a large, complex hierarchy or graph of objects.

In WebSphere ESB, the business object framework is made up of the following elements:

- ▶ Business object definition
- ▶ Business graph definition
- ▶ Business object metadata definition
- ▶ Business object services (service APIs)

A *business object definition* is the name, set of ordered attributes, properties, version number, and application-specific text that specify a type of business object. A business graph definition is the wrapper added around a simple business object or a hierarchy of business objects to provide additional capabilities, such as carrying change summary and event summary information related to the business objects in the business graph.

A *business object metadata definition* is the metadata that can be added to business object definitions to enhance their value when running on WebSphere ESB. This metadata is added to the business object's XML schema definition as well-known `xs:annotation` and `xs:appinfo` elements. Business object services are a set of capabilities that are provided on top of the basic capabilities that are provided by Service Data Objects. Examples are services such as create, copy, equality, and serialization.

Common Event Infrastructure in WebSphere ESB

The Common Event Infrastructure (CEI) is an embedded technology within WebSphere ESB that provides basic event management services. The infrastructure portion of the CEI is included as part of the underlying WebSphere Application Server capabilities in WebSphere ESB. The event emitting capabilities are additional functions of WebSphere ESB.

The CEI is the implementation of a set of APIs and infrastructure for the creation, transmission, persistence, and distribution of business, system, and network Common Base Events. A Common Base Event is a specification that is based on XML. It defines a mechanism for managing events, such as logging, tracing, management, and business events, in business enterprise applications.

CEI provides basic event-management services, including consolidating and persisting raw events from multiple, heterogeneous sources and distributing those events to event consumers. It provides functionality for generation, propagation, persistence, and consumption of events representing service component processes. A standard, XML-based format, the Common Base Event model, defines the structure of these events. Each type of event used by the server contains a number of standard fields specific to a given type of event. In certain cases, it contains an encapsulation of the business object data that is being used by the service component at a particular event point.

WebSphere ESB uses events in the CEI almost exclusively to enable service component monitoring. You must configure the CEI server if you want to use event-related functions, but after that, do not use CEI directly. Instead, use the existing services in WebSphere ESB.

WebSphere ESB includes a configured CEI Server that can be part of an existing process server or another server. You can use this server for all event-related

services. To use the server, you must first create and deploy facilities that CEI Server uses, including an event database, a messaging engine, one or more enterprise applications, and a database driver.

Mediations

Mediation modules are SCA modules that can change the format, content, or target of service requests. These modules operate on messages that are in-flight between service requesters and service providers. You can route messages to different service providers and can amend the message content or form. Mediation modules can provide functions such as message logging and error processing that are tailored to your requirements. You can change certain aspects of mediation modules from the administrative console without having to redeploy the module.

Mediation modules contain the following items:

- ▶ *Imports* define interactions between SCA modules and service providers. They allow SCA modules to call external services as though they were local. You can view mediation module imports and modify the binding.
- ▶ *Exports* define interactions between SCA modules and service requesters. They allow an SCA module to offer a service and define the external interfaces (access points) of an SCA module. You can view mediation module exports.
- ▶ *SCA components* are building blocks for SCA modules such as mediation modules. You can create and customize SCA modules and components graphically using Integration Designer. After you deploy a mediation module, you can customize certain aspects of the module from the administrative console, without having to redeploy the module.

Usually, mediation modules contain one or more of a specific type of SCA component called a *mediation flow component*. Mediation flow components define mediation flows. A mediation flow component can contain none, one, or a number of mediation primitives. WebSphere ESB supports a supplied set of mediation primitives that provide functionality for message routing and transformation. If you need additional mediation primitive flexibility, you can use the Custom mediation primitive to call custom logic.

The purpose of a mediation module that does not contain a mediation flow component is to transform service requests from one protocol to another. For example, a service request might be made using SOAP/JMS but might need transforming to SOAP/HTTP before sending.

Figure 2-5 shows a simplified example of a mediation module. This mediation module contains one mediation flow component, which contains mediation primitives.

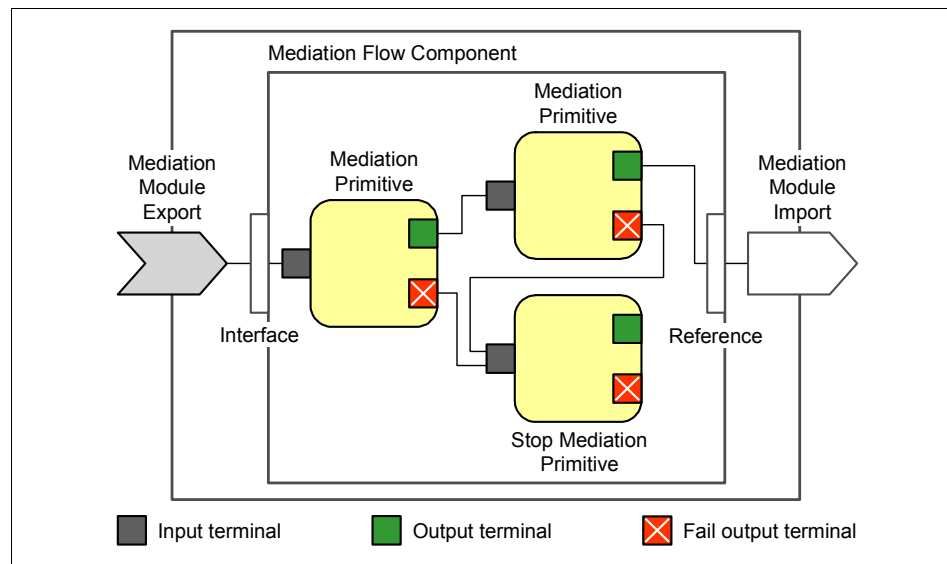


Figure 2-5 Key components of mediation modules

A mediation module contains the following key components:

- Mediation primitives have *properties*, certain of which can be displayed in the administrative console as additional properties of an SCA module. For mediation primitive properties to be visible from the WebSphere ESB administrative console, the integration developer must promote the properties.

Certain properties lend themselves to being administratively configured. Integration Designer describes these properties as *promotable* properties, because they can be promoted from the integration cycle to the administrative cycle.

Other properties are not suitable for administrative configuration, because modifying them can affect the mediation flow in such a way that the mediation module needs to be redeployed. Integration Designer lists the properties that you can choose to promote under the promoted properties of a mediation primitive.

You can use the WebSphere ESB administrative console to change the value of promoted properties without having to redeploy a mediation module, or restart the server or module.

Generally, mediation flows use property changes immediately. However, if property changes occur in a deployment manager cell, they take effect on each node as that node is synchronized. Also, mediation flows that are in-flight continue to use previous values.

Changing property values from the administrative console: You can change only property values from the administrative console. You cannot change property groups, names, or types. If you want to change property groups, names or types, you must use Integration Designer.

- A mediation module or dependent library can also define *subflows*. A subflow encapsulates a set of mediation primitives that are wired together as a reusable piece of integration logic. You can add a primitive to a mediation flow to invoke a subflow.

Messaging infrastructure

WebSphere ESB supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based, messaging infrastructure in an integrated enterprise service bus. The enterprise service capabilities that you can use for your enterprise applications provide a transport layer and mediation support to facilitate service interactions. The enterprise service bus is built around open standards and SOA. It is based on the robust Java EE infrastructure and associated platform services provided by WebSphere Application Server Network Deployment.

The *messaging infrastructure* consists of messaging or queue destination hosts, data sources and service integration buses. A service integration bus is a managed communication mechanism that supports service integration through synchronous and asynchronous messaging. A bus consists of interconnecting messaging engines that manage bus resources. It is one of the WebSphere Application Server technologies on which WebSphere ESB is based. Certain buses are created automatically for use by the system, the SCA applications that you deploy, and by other components. You can also create buses to support service integration logic or other applications, for example to support applications that act as service requesters and providers within WebSphere ESB or to link to WebSphere MQ.

Security on WebSphere ESB

Securing the WebSphere ESB environment involves enabling administrative security, enabling application security, creating profiles with security, and restricting access to critical functions to selected users. WebSphere ESB provides a runtime security infrastructure and mechanisms based on WebSphere Application Server security.

System monitoring on WebSphere ESB

You monitor events in WebSphere ESB to assess problem determination, to tune performance, and to measure the effectiveness of your business processes.

WebSphere ESB includes the following event monitoring capabilities:

- ▶ **Monitoring performance**

Performance measurements are available for service component event points and are processed through the Performance Monitoring Infrastructure (PMI) and the IBM Tivoli® Performance Viewer.

You can monitor specific performance measurements for a given event, such as the number of times the event is invoked or the length of time it takes for that event to complete from start to finish. You can also monitor events and later view their contents, either by viewing the events in a log file or by querying the events stored on the event database. In both cases, you can temporarily specify an event point or points to monitor to spot problems with the application logic or with system performance.

- ▶ **Monitoring service component events**

WebSphere ESB monitoring can capture the data in a service component at a certain event point. These events are formatted in a standard called the Common Base Event. You can have the WebSphere ESB publish these events to the logging facilities, or you can use the more versatile monitoring capabilities of a Common Event Infrastructure server database to store and analyze these events.

2.2.3 IBM Integration Designer

IBM Integration Designer is the authoring environment for artifacts to be created and then deployed to WebSphere ESB. Integration Designer provides editors and aids to help developers create automated processes and services. It is available as a component in IBM Business Process Manager Advanced or as a stand-alone toolset for other uses.

Integration Designer is designed as a complete integration development environment for those who are building integrated applications. Integrated applications are not simple. They can call applications on Enterprise Information Systems (EIS), involve business processes across departments or enterprises, and invoke applications (locally or remotely) that are written in a variety of languages and that are running on a variety of operating systems.

The components are created and assembled into other integrated applications (that is, applications created from a set of components) through visual editors. The visual editors present a layer of abstraction between the components and their implementations. A developer using the tools can assemble an integrated

application without detailed knowledge of the underlying implementation of each component.

Integration Designer tools are based on an SOA. Components are services, and an integrated application that involves many components is also a service. The services that are created comply to the leading, industry-wide standards. In the Integration Designer paradigm, components are assembled in modules. Imports and exports are used to share data between modules. Artifacts placed in a library can be shared among modules.

Mediation modules with mediation flows, business object maps, interface maps, relationships, and connectivity using adapters are authored and assembled using Integration Designer for deployment to WebSphere ESB.

2.3 WebSphere ESB Registry Edition dynamic ESB usage patterns

SOA patterns have many requirements that are best served by an integrated ESB and service registry. In addition to these requirements, WebSphere ESB Registry Edition supports a number of ESB pattern where WSRR and WebSphere ESB combine to provides a dynamic, flexible, and governed ESB platform. The following sections provide an overview of three such major patterns.

2.3.1 Dynamic service gateway pattern

Many enterprises want to access multiple services through one generic interface. Service gateways act as proxies to services and, therefore, allow you to access multiple services from one address. In addition, service gateways encapsulate transformations, routing, and common processing.

The dynamic service gateway pattern creates a service gateway using a mediation flow in WebSphere ESB that proxies multiple services. You retrieve endpoints from these services at run time from a number of sources, including WSRR. This pattern provides a flexible and yet well-governed service selection mechanism. The service is managed and governed through its complete life cycle in WSRR. The mediation is implemented and exposed on WebSphere ESB, and the endpoint is retrieved by the mediation from WSRR during run time.

2.3.2 Mediation policy patterns

Many enterprises want to control their service interactions dynamically using context information. Existing services need to be accessed in constantly changing ways. You need to be able to govern service interactions, so that when circumstances change you can adjust the interactions rather than change the applications. The mediation policy can be useful either to control the logic inside the mediation module or to select the appropriate target service at the end of the mediation module.

In cases when the need is to control the logic inside the WebSphere ESB mediation module, the SCA module receives an incoming message at run time and queries the WSRR instance that was loaded with the mediation module and its mediation policies. Based on the contents of the incoming message, the appropriate mediation policy is decided upon and applied, resulting in policy-based mediation logic.

In cases when the need is to select the appropriate target service, the SCA module receives a message at run time and queries the WSRR instance that was loaded with the module, the WSDL documents, and their mediation policies. WSRR returns the mediation policy that is associated with the service that is called. The policy is evaluated, and the appropriate target service is invoked.

2.3.3 Service level agreement-related patterns

Another crucial mechanism of control can be by exploiting service level agreements (SLAs). WSRR stores SLAs as part of its governance enablement profile. A consumer service representation in WSRR can have an SLA that describes the contract to a provider service definition. WebSphere ESB can verify this SLA to determine whether a consumer is entitled to invoke a service. Additional metadata stored in the SLA describes the agreed parameters in which the consuming service is allowed to operate, for example the level of service, minimum and maximum messages per day, and so on.

In cases when a SLA is created between a service consumer and provider, WebSphere ESB mediation decides whether to proceed with the invocation after performing a query to WSRR on whether there is an active SLA between the two parties that referenced the required endpoint.

In other cases, the SLA can be associated with a number of available endpoints so that the WebSphere ESB mediation must select between them based on various selection criteria, including the SLA. The mediation queries WSRR to evaluate and find an endpoint that matches the criteria.



Part 2

WebSphere Enterprise Service Bus Registry Edition topologies



Topology overview

In this chapter, we provide an overview of the most common types of topologies that are related to WebSphere ESB Registry Edition.

Topology note: The complete architecture of WebSphere ESB Registry Edition is based on WebSphere Application Server Network Deployment. Therefore, all topologies that are applicable to WebSphere Application Server Network Deployment also apply to WebSphere ESB Registry Edition. For the purposes of our discussion, we assume that you understand these concepts. You can find more information in *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688.

Although many nonfunctional requirements determine which topology you need to use, in this chapter we focus only on the workload and its related nonfunctional requirements.

3.1 Workload concepts

Conceptually, a system's *workload* corresponds to the system's capability to process a certain number of events in a given time window. Performance is related to the response time but depends on the workload. Workload depends on the following factors:

- ▶ Number of users
- ▶ Number of sessions (HTTP sessions, EJB sessions, and so forth)
- ▶ Concurrency
- ▶ Number of accesses in a time window
- ▶ Response time
- ▶ Session time per user
- ▶ Resource capacity (disk, memory, CPU, and so forth)

Instances require a set of resources to support the workload that is generated by the environment. For the purposes of our discussion, we define these workloads as follows:

- ▶ Low Workload

With a Low Workload topology, no concurrency is generated and not many users are connected. The Java virtual machine (JVM) does not process more than two user threads at a time. User requirements do not need more resources than the standard product configuration.

- ▶ High Workload

With a High Workload topology, concurrency is expected. User requirements might need more resources than the standard product configuration.

Terminology note: The terminology that we use in this chapter is relevant to the workload. Thus, the term *instance* for all WebSphere products is always related to a JVM instance that runs below WebSphere Application Server Network Deployment. For DB2, it is the process that runs the DB2 instance (by default, listening on port 50000).

3.2 WebSphere ESB Registry Edition Low Workload topology

The Low Workload topology offers the best option for development environments and, sometimes, also for test environments that are used for integration and user acceptance tests. This topology is not the best choice for a production environment, even when a really low workload is estimated. However, it can be

considered an alternative as a startup environment. If scalability is required, the Low Workload topology is not satisfactory.

3.2.1 Basic Low Workload topology

The development environment for WebSphere ESB Registry Edition includes IBM Integration Designer for developing artifacts, such as mediation flows or SCA components, and a stand-alone installation of the runtime environment for testing these artifacts. WSRR Studio is also part of the development environment but, as described in 4.5.3, “Installing and configuring WSRR Studio” on page 124, is not essential for development. You can install all products on the same physical system as long as the software and the hardware requirements are met.

The simplest development environment installs the complete WebSphere ESB Registry Edition runtime environment in only one underlying WebSphere Application Server instance. WSRR and WebSphere ESB are deployed on top of this WebSphere Application Server instance. The database schemas for WSRR and WebSphere ESB are in a single database only.

Figure 3-1 shows the topology for the simplest development environment.

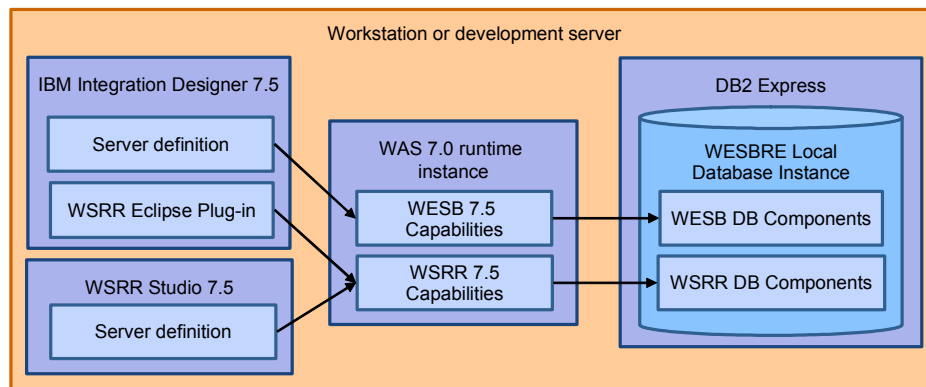


Figure 3-1 Development environment for WebSphere ESB Registry Edition

The development environments can also be separated, and the database and runtime environment can be located on another system. However, this configuration is not ideal if local resources can support such a workload on the local system because it creates more dependencies on the context.

For example, if the local system does not comply with memory requirements for WebSphere ESB Registry Edition but complies with memory requirements for WebSphere ESB and its database only, you need another system to complement

the entire solution. In this example, the developer must manage two systems and the connection between them. Alternatively, if the system has enough memory for the complete WebSphere ESB Registry Edition solution, using only one system is a leading practice to avoid dependencies with other resources (systems, networks, and so forth).

3.2.2 Advanced Low Workload topology

A more complex Low Workload topology, as illustrated in Figure 3-2, is an extension of the basic Low Workload topology. In this topology, two separate instances of the WebSphere ESB and the WSRR are used for the run time. Thus, each product has its own underlying WebSphere Application Server instance. This configuration needs more resources than the basic Low Workload topology, but it can be used as a development environment.

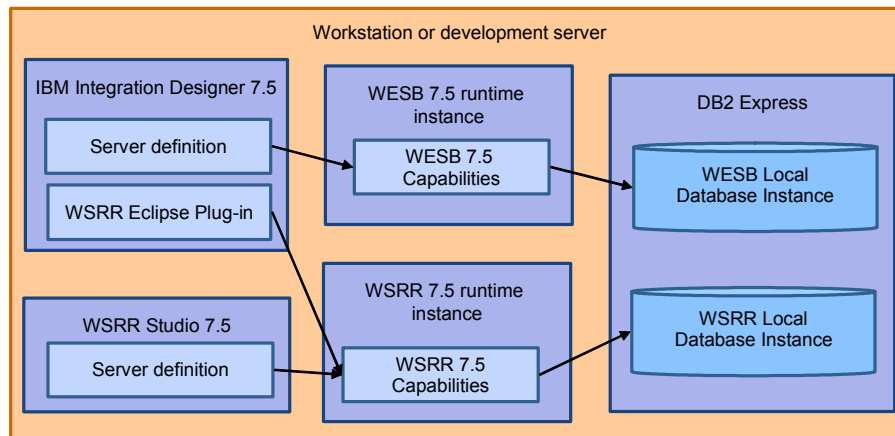


Figure 3-2 Advanced Low Workload topology

With this topology, there are two different WebSphere Application Server instances, each with an independent set of configurations. These configurations provide a more appropriate topology for development because at the JVM level, the configuration is similar to a production configuration and issues can be detected before going to the test environment. For example, if a memory problem is generated by a mediation flow, this topology provides a way to detect the component that is causing the issue because WebSphere ESB is alone. The basic Low Workload topology has only one JVM, so it can be more difficult to determine which component is consuming memory.

Splitting the database is another alternative and has similar advantages and disadvantages as the topology that we describe here. Splitting the database

consumes more resources but adds more independence for configuration. Combinations for the basic and advanced topology are as follows:

- ▶ One instance of the WebSphere ESB Registry Edition runtime environment (as shown in Figure 3-1) combined with two databases (WSRR database separated from WebSphere ESB database) but using the same DB2 instance as shown in Figure 3-2
- ▶ One instance of WebSphere ESB and a separate instance of WSRR (as shown in Figure 3-2) using only one database as shown in Figure 3-1

This environment is not supported for DB2 Express; therefore, an upper-level DB2 product, such as DB2 Workgroup Server Edition, is required.

3.3 WebSphere ESB Registry Edition High Workload topology

In this section, we describe how to deploy WebSphere ESB Registry Edition topologies in a production environment. The topologies that we describe are workload-dependent only. We do not discuss alternatives to defining topologies based on other nonfunctional requirements, such as disaster recovery, security, maintainability, performance, or administration. We do discuss improvements to allow scalability and high availability.

3.3.1 WSRR deployment topologies

In this section, we describe different methods of deploying WSRR, depending on the type of production environments. All the deployment environments that we describe are based on the principle that one instance of the registry is dedicated to keeping a governance view of the service metadata, and that another instance stores the metadata that is used by the runtime services in production.

This method of deploying WSRR is the starting point for supporting a high workload in a production environment. During the development life cycle, users often interact with the registry. These interactions generate workload. With this approach, most of the changes in service metadata during the service development life cycle do not impact the runtime operation.

Promotion can be achieved using the synchronous promotion capability or manual file promotion. When promoting a service that is ready to be used in the production environment, the service metadata must be moved from the governance registry to the runtime registry. Only at this point do both instances interact, thus reducing the workload into the runtime registry. Manual file

promotion does not require WSRR instances to interact and may be desirable due to firewall issues or existing IT policies.

Pilot deployment is the simplest way to deploy WSRR with high workload requirements. This deployment is described in Figure 3-3. In Figure 3-3, synchronous replication is implemented through an EJB.

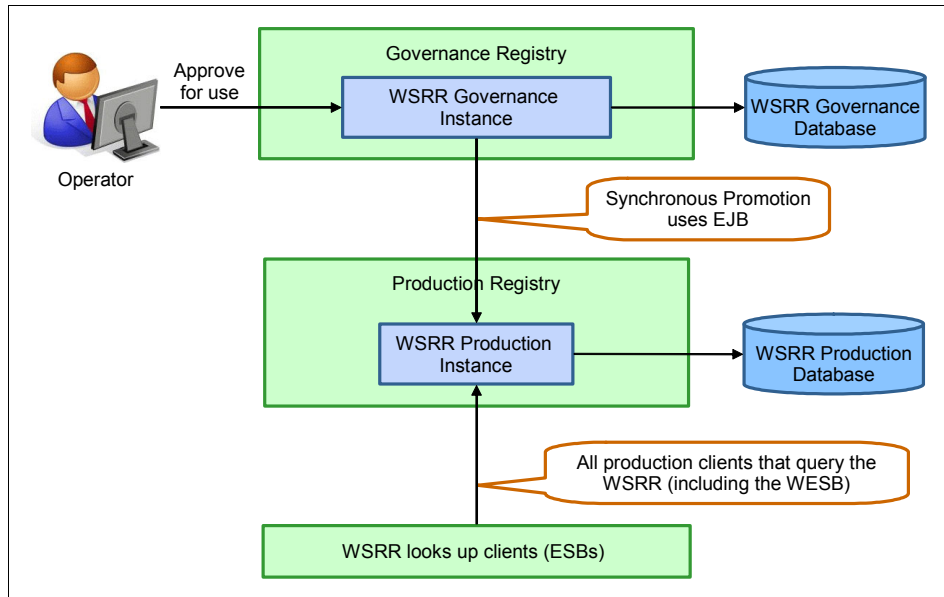


Figure 3-3 Promoting a service into production

Full production deployment

A *full production deployment* extends a pilot deployment by adding multiple registries. The number of registries depends on the quantity of stages that are included in the development and testing life cycles.

According to various development methodologies, such as IBM UMF Methodology and the Governance Enablement Profile, testing is a crucial component when changing a service from one stage to another. Therefore, the service metadata for the development and testing stages must be stored in separate registries to isolate each stage and to control the service before it goes into production.

Use the following stages for testing with the corresponding registries for each stage, as illustrated in Figure 3-4:

1. In the *integration* stage, testing ensures that the service functions correctly. However, related components might not be available to the same department that is doing the integration testing.
2. In the *acceptance* stage, testing ensures that the service and all the related components meet the desired business state.
3. In the *preproduction* stage, the target is to simulate the behavior of a production system. Testing in this environment reduces the risk of deploying issues when promoting the service into the production environment.
4. Finally the last stage is going into *production*.

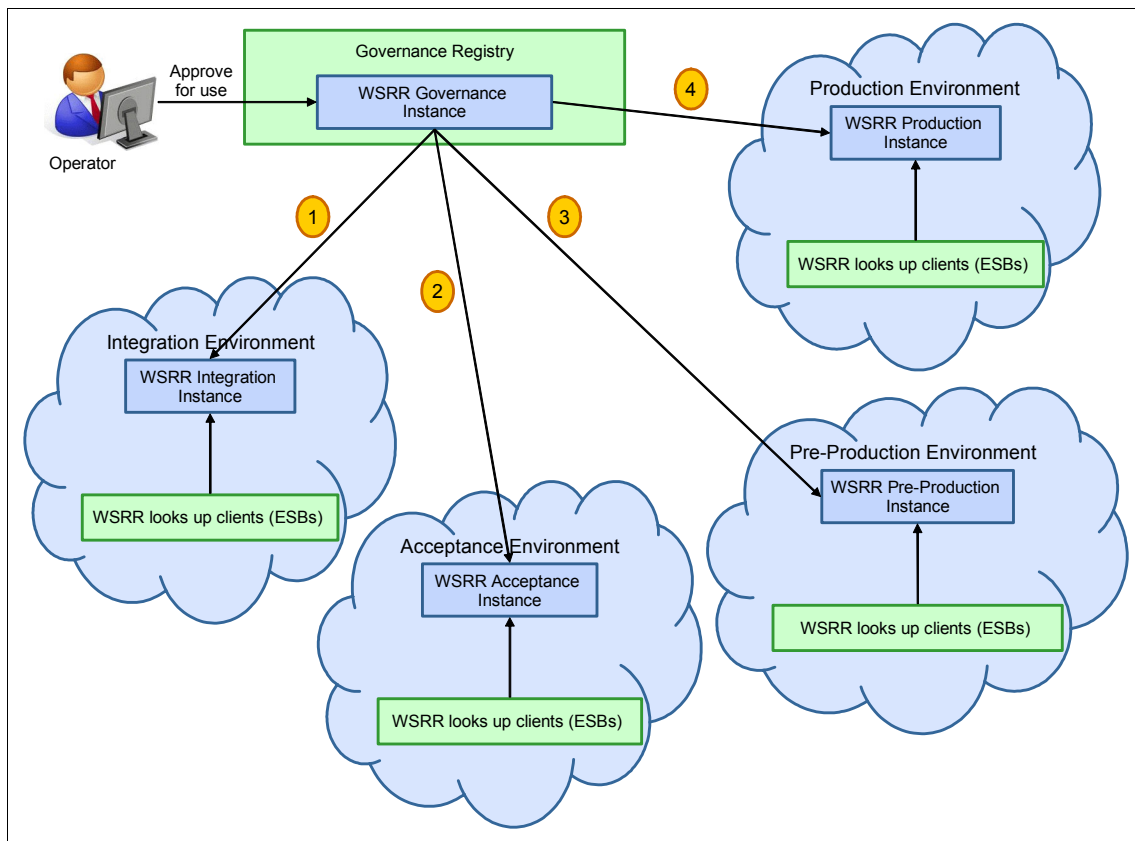


Figure 3-4 Full production deployment environment

Organizational federation deployment

Organizational federation deployment is a special way of deploying WSRR and applies when two or more service domains need to be centralized. A service domain can be identified at the following scopes:

- ▶ Big business areas in the same company, for example a social security company

One service domain can be identified for the business area that is responsible for collecting money, and another service domain can be identified for the area that provides social services.
- ▶ An acquisition or merger

When an acquisition or merger occurs, service domains can be separated along organizational or even geographical lines.

In general, many service registries and repositories must be integrated to achieve an overall view of all the services in a company.

3.3.2 WebSphere ESB deployment topologies

WebSphere ESB and many of the products based on it, such as WebSphere Process Server, can be deployed using *patterns*. These patterns are based on the most commonly used topologies. For example, a deployment environment pattern specifies the constraints and requirements of the components and resources that are involved in a deployment environment. These patterns are designed on the nonfunctional requirements and represent well-known, tested, and recommended topologies with component configurations that work together. Using these patterns ensures a properly working deployment environment.

The configuration rules for these deployment environment patterns enable an IT architect and the IT administrator to rapidly instantiate architectural decisions that are already provided by the pattern, such as workload management. Each of the supplied deployment environment patterns addresses a specific set of requirements. Most requirement sets can be met using one of the following patterns:

- ▶ Single Cluster
- ▶ Remote Messaging
- ▶ Remote Messaging and Remote Support
- ▶ Remote Messaging, Support, and Web Application

Single Cluster

The Single Cluster pattern is the simplest pattern. It defines one application deployment target cluster that contains all deployment components. It is suitable

for scenarios that focus on running applications with synchronous invocations. With this pattern, keep any messaging requirements to a minimum.

Service Component Architecture (SCA) internal asynchronous invocations, the Java Message Service (JMS), and MQ messaging bindings do not support multiple messaging engines in the same cluster. If your modules require any of these mechanisms, choose one of the other patterns. The messaging infrastructure in those patterns is based in a separate cluster from the application deployment target cluster.

The Single Cluster pattern consists of the following components:

- ▶ SCA application bus members
- ▶ SCA system bus members
- ▶ Common Event Infrastructure (CEI) bus members
- ▶ CEI server
- ▶ Application deployment target

Remote Messaging

The Remote Messaging pattern defines one cluster for application deployment and one remote cluster for the messaging infrastructure. The CEI application and other supporting applications are configured on the application deployment target cluster.

The Remote Messaging pattern provides a separate cluster for the messaging role. This pattern is suitable for scenarios involving asynchronous invocations, because the cluster can be scaled for this load. The Remote Messaging pattern defines two clusters:

- ▶ Support infrastructure and application deployment target cluster
 - CEI server application
 - Application deployment target
- ▶ Remote messaging cluster
 - SCA application bus members
 - SCA system bus members
 - CEI bus members

Remote Messaging and Remote Support

The Remote Messaging and Remote Support pattern is the default pattern for WebSphere ESB. With this three-cluster pattern, resources are allocated to the cluster that handles the highest workload. This pattern is the most flexible and versatile and is preferred by most users.

This pattern consists of the following clusters:

- ▶ Remote messaging infrastructure
 - SCA application bus members
 - SCA system bus members
 - CEI bus members
- ▶ Remote support infrastructure
 - CEI server application
- ▶ Application deployment
 - Application deployment target

Remote Messaging, Support, and Web Application

This four-cluster pattern is similar to the Remote Messaging and Remote Support pattern, except that the supporting web applications reside on their own cluster. The Remote Messaging, Support, and Web Application pattern defines the following clusters:

- ▶ Remote messaging infrastructure
 - SCA application bus members
 - SCA system bus members
 - CEI bus members
- ▶ Remote support infrastructure
 - CEI server application
- ▶ Application deployment
 - Application deployment target
- ▶ Remote web application infrastructure
 - Web application deployment target (user interfaces, business space)
 - REST services that are related web applications

3.3.3 Main High Workload topology for WebSphere ESB Registry Edition

Workload into a complex system can be generated at many levels. However, when defining the topology for WebSphere ESB Registry Edition, the following entry points into the product are the most critical:

- ▶ The amount of incoming requests to the WebSphere ESB is the sum of all requests that are generated by service consumers that want to communicate over the ESB.

- The amount of incoming requests to the WSRR, for which most of the income is generated by the WebSphere ESB but also other clients such as stand-alone J2EE applications or other ESBs, can query information from the registry.

Scalability

This section discuss various improvements to the previous topologies regarding scalability. Scalability must be designed at the beginning of a solution, especially if you are using WebSphere ESB Registry Edition as the solution for a company that expects to grow quickly.

All products included in WebSphere ESB Registry Edition are based on WebSphere Application Server Network Deployment; therefore, we can achieve a scalable topology using the features that are provided with WebSphere Application Server Network Deployment.

In the previous topology, depending on which component of WebSphere ESB Registry Edition was forced to scale, either WebSphere ESB or WSRR clients need to change because new servers have to be added to the solution. However, if we use the clustering capabilities of WebSphere Application Server Network Deployment, this change is not needed because of IBM HTTP Server, as shown in Figure 3-5.

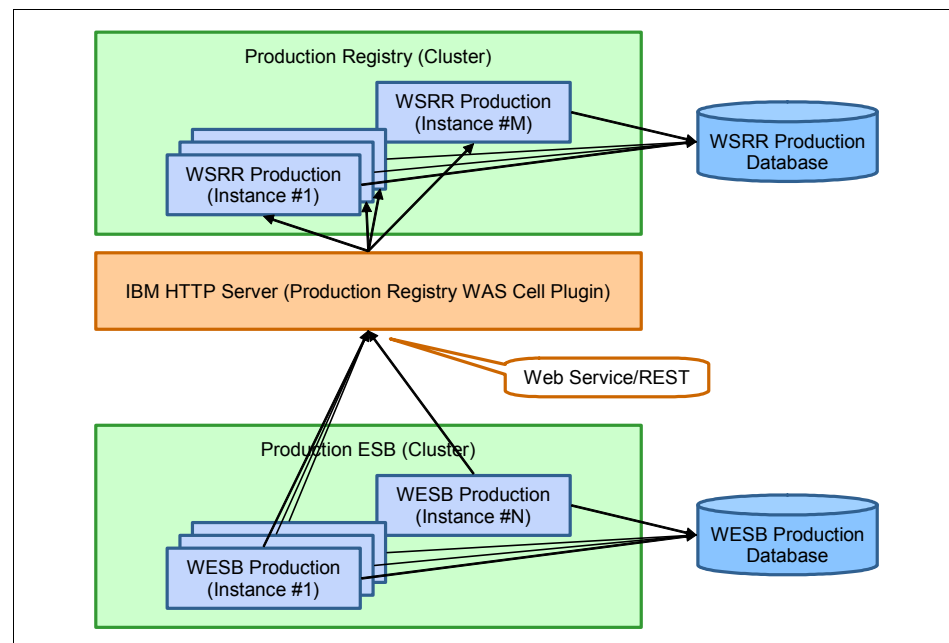


Figure 3-5 A scalable topology for WebSphere ESB Registry Edition

The IBM HTTP Server in this scenario is used as a web server for static content (HTML pages, images, and so forth) and also as a proxy server for all HTTP requests. IBM HTTP Server is provided by WebSphere Application Server Network Deployment with the web server plug-in for WebSphere Application Server. This plug-in keeps the clients unmodified, even when the topology scales up. IBM HTTP Server redirects all incoming HTTP/HTTPS requests automatically to newly added servers when the plug-in file is updated.

As shown in Figure 3-5, clients do not point the URL to servers in the WebSphere ESB or WSRR cluster. Instead, the URL points to only IBM HTTP Server. When new servers are added to the topology, the change in the plug-in file and its update to the IBM HTTP Server can be accomplished easily. Figure 3-5 does not illustrate IBM HTTP Server between service consumers and the ESB itself, but the same strategy applies in that case.

Refer to *WebSphere Application Server V7: Concepts, Planning and Design*, SG24-7708, for more details about IBM HTTP Server and WebSphere Application Server Network Deployment.

High availability and balancing

This book does not discuss which strategy is the best strategy for implementing a high availability solution for the overall infrastructure. Operating system approaches, such as IBM HACMP, or other providers' solutions for databases, such as Oracle Real Application Clusters (Oracle RAC), can be included to improve the overall infrastructure, but we do not discuss these topics in this book. Our discussion focuses on WebSphere solutions for high availability and balancing.

This section discusses improvements to the topology that we described in “Scalability” on page 57. These improvements allow the solution to increase availability and to optimize and take advantage of all available resources through workload balancing.

With the scalability topology, the IBM HTTP Server instance fails when WebSphere ESB cannot receive a response from the registry. Therefore, no high availability exists. We improve the availability by introducing the Load Balancer between WebSphere ESB and IBM HTTP Server, as shown in Figure 3-6, which solves this issue. Alternatively the Load Balancer can sit in front of the HTTP server (as demonstrated in Figure 3-7).

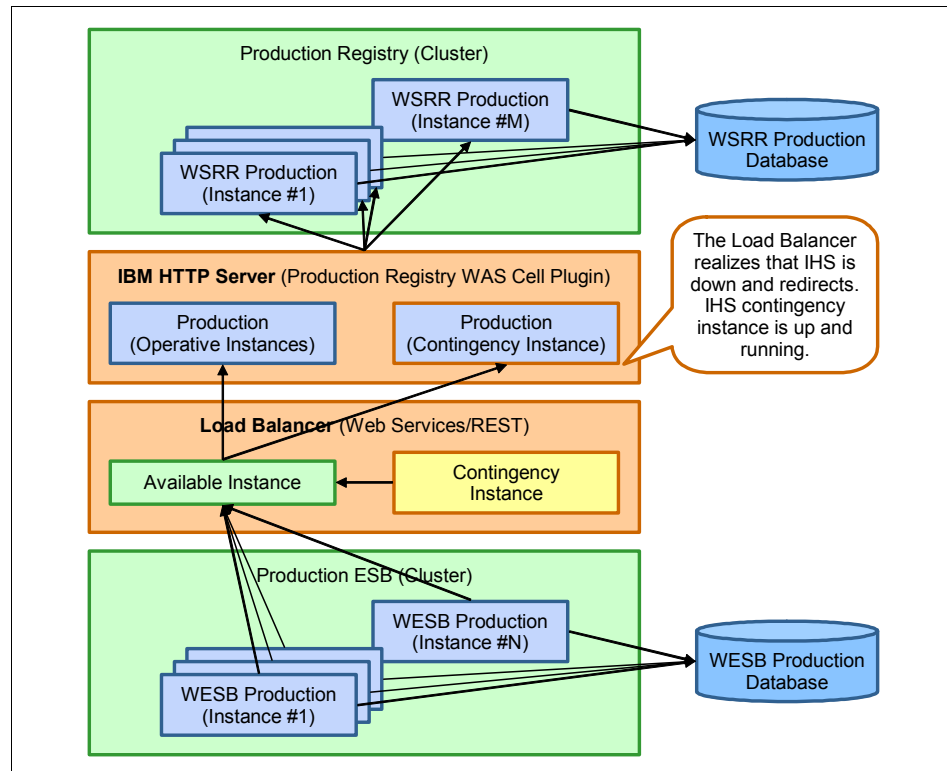


Figure 3-6 A high availability topology with workload balancing

The Load Balancer is one of the Edge Components for WebSphere Application Server. As its name implies, it balances the workload that is generated by the service consumers among the requested targets. The WebSphere ESB server that uses web services (HTTP/HTTPS) for lookups can then communicate with the IBM HTTP Server instances through the Load Balancer instance.

We can also work without the IBM HTTP Server instance to communicate between WebSphere ESB and WSRR using the Load Balancer. However, for user interfaces such as Business Space, keep IBM HTTP Server and use the Web Server plug-in for WebSphere Application Server to allow some static content to reside at the web server level.

Figure 3-7 illustrates a sample topology of a complete production environment of WebSphere ESB Registry Edition with high availability and user interaction.

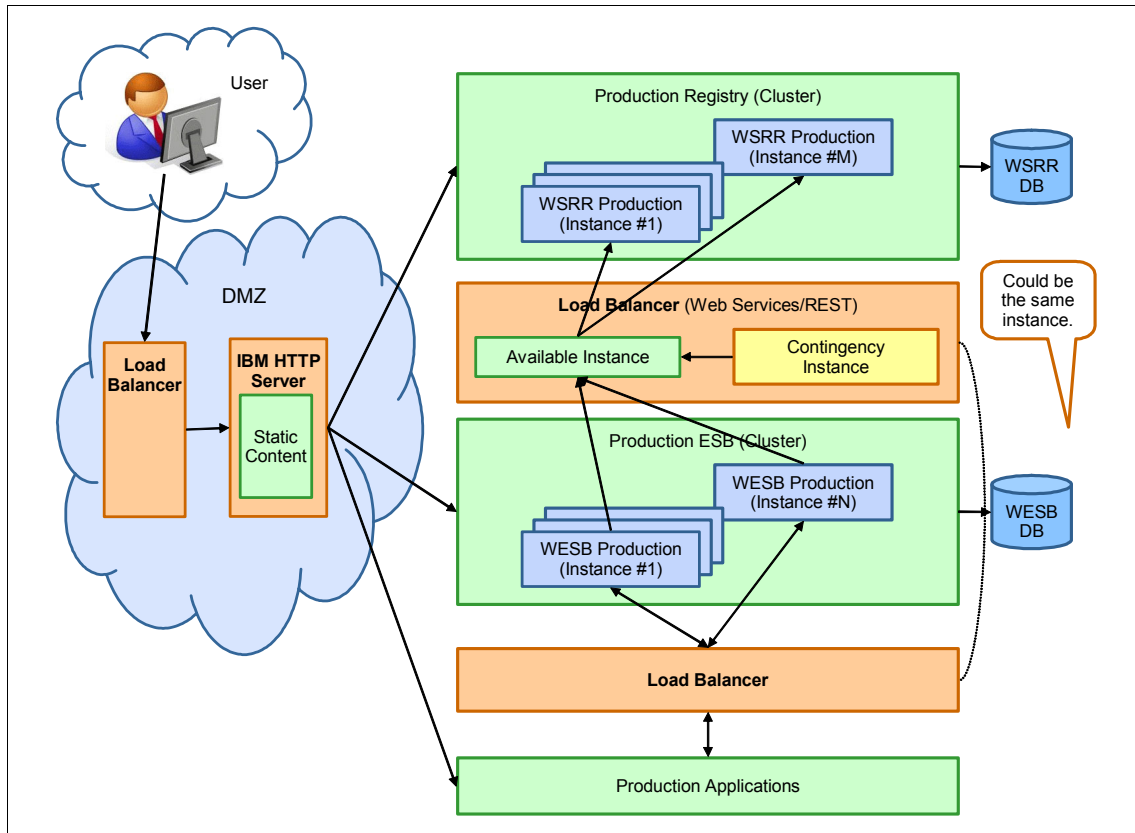


Figure 3-7 A complete topology for production environments

In Figure 3-7, the arrows represent any type of request from the consumer to the provider. For example a Production Application sends requests to the Load Balancer (for consuming services from the ESB) but also receives requests from the Load Balancer (applications are also services providers to the ESB).

The most important feature of the Load Balancer is the capability of having contingency instances (or backup instances) in standby mode that monitor the operative instance. When this monitor detects that the operative instance is down, it takes control of the balancing automatically using the same identification attributes, such as IP or host name. Therefore, the failure of the operative instance is transparent to the consumers.

Refer to the following resources for more details about this product and high availability:

- ▶ *Patterns for the Edge of Network*, TIPS0062
<http://www.redbooks.ibm.com/abstracts/tips0062.html?Open>
- ▶ IBM WebSphere Application Server Information Center
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.edge.doc/lb/info/ae/welcome_overview.html
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
<http://www.redbooks.ibm.com/abstracts/sg246688.html?Open>

3.4 Sizing a WebSphere ESB Registry Edition solution

Sizing a solution depends on several factors, such as:

- ▶ Base infrastructure
- ▶ Integration architecture
- ▶ Nonfunctional requirements
- ▶ Architectural decisions
- ▶ Mediation flows design
- ▶ Message size

You determine the size of a solution by how many resources are needed to support the overall architecture. In this discussion, *resources* can be anything from the number of servers or the amount of CPU to the memory capacity and disk space.

In this section, we discuss how to size a WebSphere ESB Registry Edition solution appropriately.

Version note: The workload scenarios that we developed and describe in this section do not apply for versions prior to 7.5.

3.4.1 Sizing WebSphere ESB

In this section, we describe the mediation flows that are designed, the adapters that are used, and other factors to clarify how to perform sizing for WebSphere ESB.

Mediation flow workload scenarios

Different scenarios can apply to a solution. In this section, we describe possible workload mediations scenarios and describe samples of usage patterns.

Mediation scenarios for sizing are analyzed and improved to better reflect supplied patterns and customer usage of the product and to list the types of mediation that will be in the sizing tools in the future. This comparison is based on multiple scenarios that are defined to compare workload. These mediation scenarios are ordered by the associated cost in the workload, from the least expensive to the most expensive:

- ▶ Service Gateway Routing Mediation
- ▶ Composite Mediation
- ▶ Protocol Conversion Mediation
- ▶ Data Warehouse Mediation
- ▶ Chained Mediation
- ▶ Aggregation Mediation

Service Gateway Routing Mediation

The Service Gateway Routing Mediation applies the Proxy Service Gateway pattern (available in IBM Integration Designer). In this pattern, the workload depends on whether the routing decision is in WebSphere ESB or WSRR. For our example, the workload is in WebSphere ESB and uses internal routing tables.

Routing can be achieved using either header-based or body-based routing, according to the associated cost in the workload. Header-based routing is less expensive than body-based routing.

Figure 3-8 shows a sample of a request flow using this mediation.

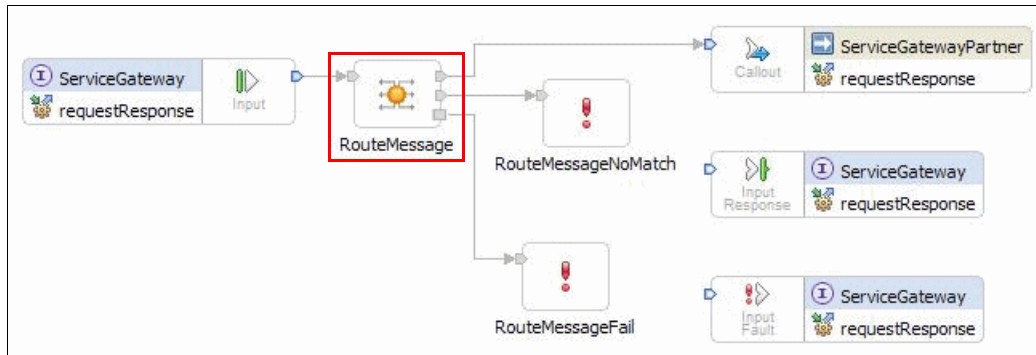


Figure 3-8 Service Gateway Routing Mediation sample

This mediation uses the Gateway Endpoint Lookup primitive to provide a single access point for multiple services and to perform common processing on all inbound and outbound messages.

This type of processing might be used to process a car insurance quotation, for example to obtain information that pertains to the make and model of the vehicle. The mediation can provide a single access point. It uses the dynamic endpoint lookup functionality to locate the appropriate service to invoke, depending on the car brand. This scenario further allows for dynamic addition and removal of target services as new makes or models of vehicles become available without the need to modify the deployed application.

Composite Mediation

The Composite Mediation is a mediation that composes multiple service interaction primitive invocations in a flow. It can also compose other pattern directed flows, but for the workload perspective we do not include compositions of other described flows. Otherwise, the associated cost of the mediation depends on the composed mediation.

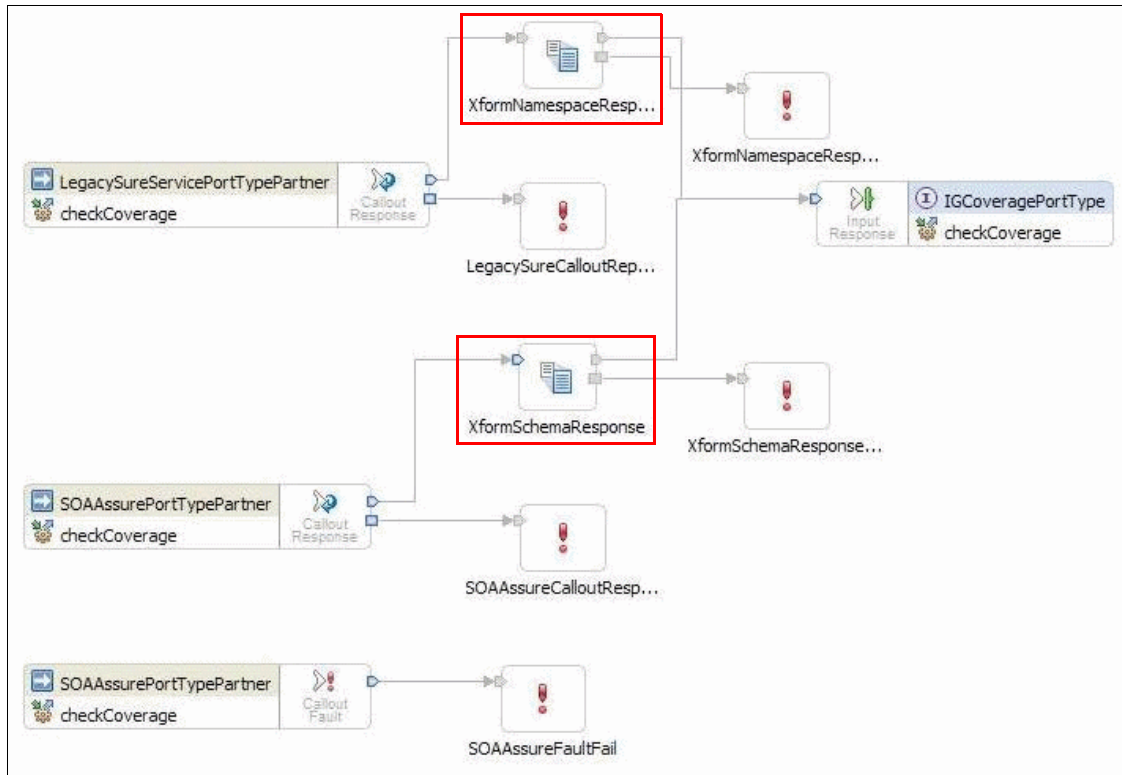


Figure 3-10 Composite Mediation response flow sample

This mediation uses the Message Filter primitive to provide static endpoint resolution based on information that is stored within the body of the incoming message. It also uses the XSLT Transform primitive to apply an appropriate transformation to the incoming message to match the expected schema of the target service.

This type of processing might be used to process a car insurance quotation by a company that has a choice of multiple, functionally equivalent, back-end services. For example, if a takeover of an existing company occurs, the company can use the acquired services. An authorization is performed, and a log is made of the request. The request is then formatted appropriately, and a particular back-end service is invoked based on whether the quotation pertains to a car, van, or motorcycle.

Protocol Conversion Mediation

The Protocol Conversion Mediation is used when protocol conversion is needed.

In our sample, the application exploits a JMS export and Web Services import. This model allows for an integration with existing JMS systems and exploits the use of Web Services. The SCA assembly diagram explicitly indicates imports and exports with the associated protocol, as shown in Figure 3-11.

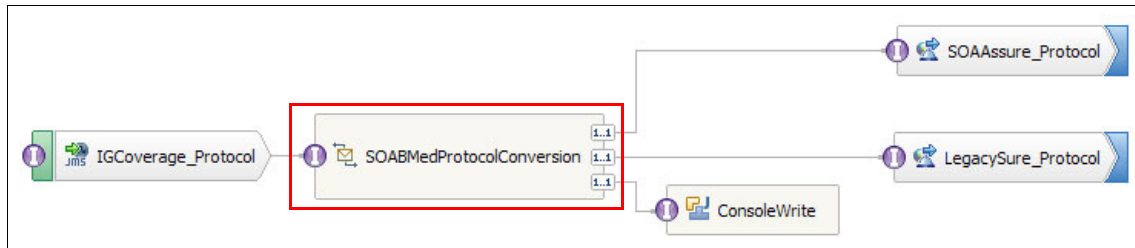


Figure 3-11 SCA Assembly diagram sample for a Protocol Conversion

This type of processing might be used to process a car insurance quotation by a company that has a choice of multiple, functionally-equivalent, back-end services. For example, if a company takes over an existing company, they can use the acquired services. An authorization is performed, and a log is made of the request. The request is then formatted appropriately, and a particular back-end service is invoked based on whether the quotation pertains to a car, van, or motorcycle.

Data Warehouse Mediation

The Data Warehouse Mediation is used to store information in databases that are used for business intelligence (Data Warehousing, OLAP, Data Mining, and so forth) or to store information for another flow or application to use. However, the latter use of this data is not important for the workload, but the archiving task is.

Our sample of this mediation implements the Message Logger primitive to perform archiving of data into a database, as shown in Figure 3-12.

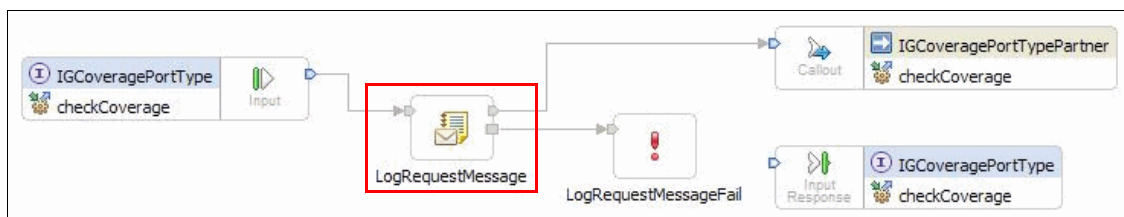


Figure 3-12 Data Warehouse Mediation sample

This type of processing might be used to store sales data for analysis at a later date. The message data is stored in a way that makes it easy to select records

for specified times. Storing a time stamp with the message content makes it possible to retrieve records between specific dates or times for weekly, monthly, or time of day comparisons. One such usage of this data might be to determine peak periods at which extra resources might be required or periods of low throughput when resources can be conserved to minimize running costs.

Chained Mediation

The Chained Mediation is another alternative to implementing the same service interaction composition that we described in “Composite Mediation” on page 63 but a higher level of integration. (The service that is called is not in the same module, and therefore must be accessed using SCA components. This scenario is more expensive in workload than previous scenarios but provides more flexibility.

The Chained Mediation and Composite Mediation can be mixed, but the idea from the workload pattern perspective is that one mediation does not include the other. The previous samples that we described for the Composite Mediation represent the same functions in a Chained Mediation scenario. The samples are divided into two request flows to show the integration on an upper level. The mediation flow level is lower than SCA component level.

The first sample is related to the request, as shown in Figure 3-13, where there is a service callout that solves the same functions illustrated in Figure 3-9 on page 64 from the RouteProvider primitive.

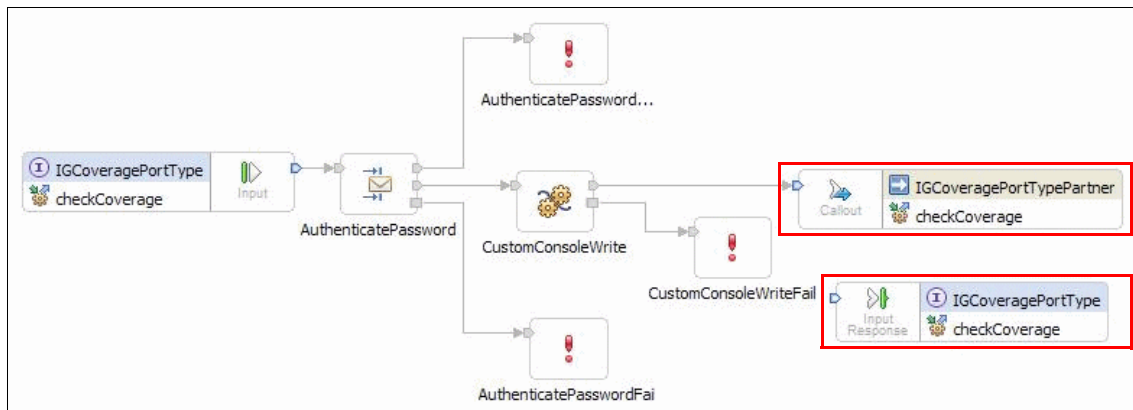


Figure 3-13 Chained Mediation sample, Part 1

The second sample, shown in Figure 3-14, is also related to the request and solves the same functions illustrated in Figure 3-9 on page 64 from the RouteProvider primitive to complement the previous request flow.

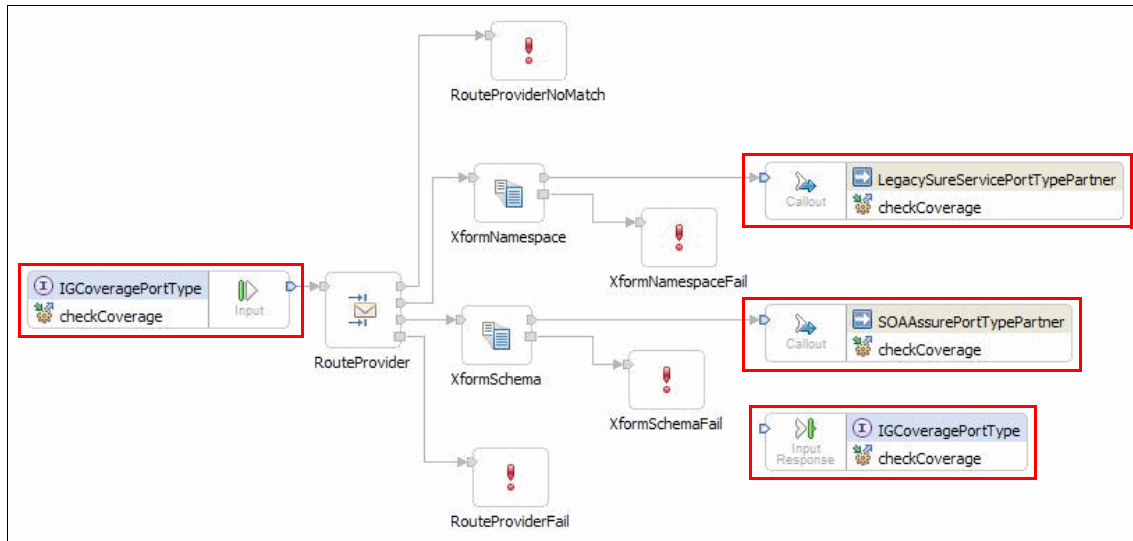


Figure 3-14 Chained Mediation sample, Part 2

This sample represents another flow but complements the first flow. When integrated, both flows have the same functions as the sample illustrated in the request flow in Figure 3-9 on page 64. The SCA assembly diagram must show the chained integration of these two flows, as illustrated in Figure 3-15.

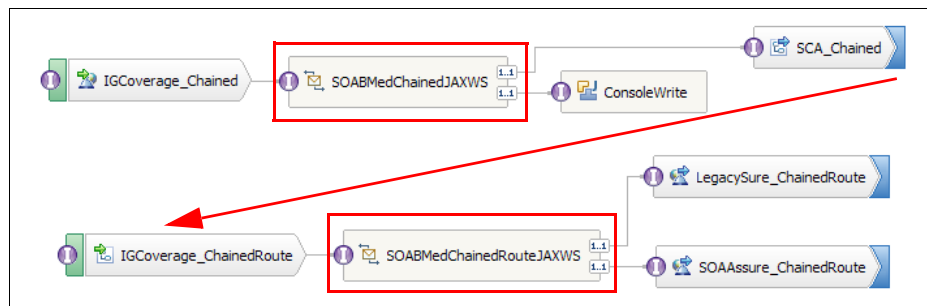


Figure 3-15 SCA assembly diagram sample for a Chained Mediation

A similar approach must be performed with responses flows.

This Chained Mediation uses the Message Filter primitive to provide static endpoint resolution based on information that is stored within the body of the incoming message. It also uses the XSLT Transform primitive to apply an appropriate transformation to the incoming message to match the expected schema of the target service. This component-based structure with two separate modules, connecting the modules using SCA bindings, allows the

implementation of additional modules, which reuses the existing mediation logic of the second module.

This type of processing is the same that we described for the Composite Mediation with the added ability to deploy additional front-end modules to allow for security processes to be enforced. For example, with our previous example, whether the vehicle quotation is being requested internally or externally determines if this mediation is appropriate. Therefore, this mediation flow is more flexible than the Composite Mediation, but it adds more workload.

Aggregation Mediation

The Aggregation Mediation applies aggregation techniques to the message flow and is the most expensive mediation. It divides the input message into pieces to process each piece separately and to aggregate these results to build the output message. Figure illustrates a sample of this mediation.

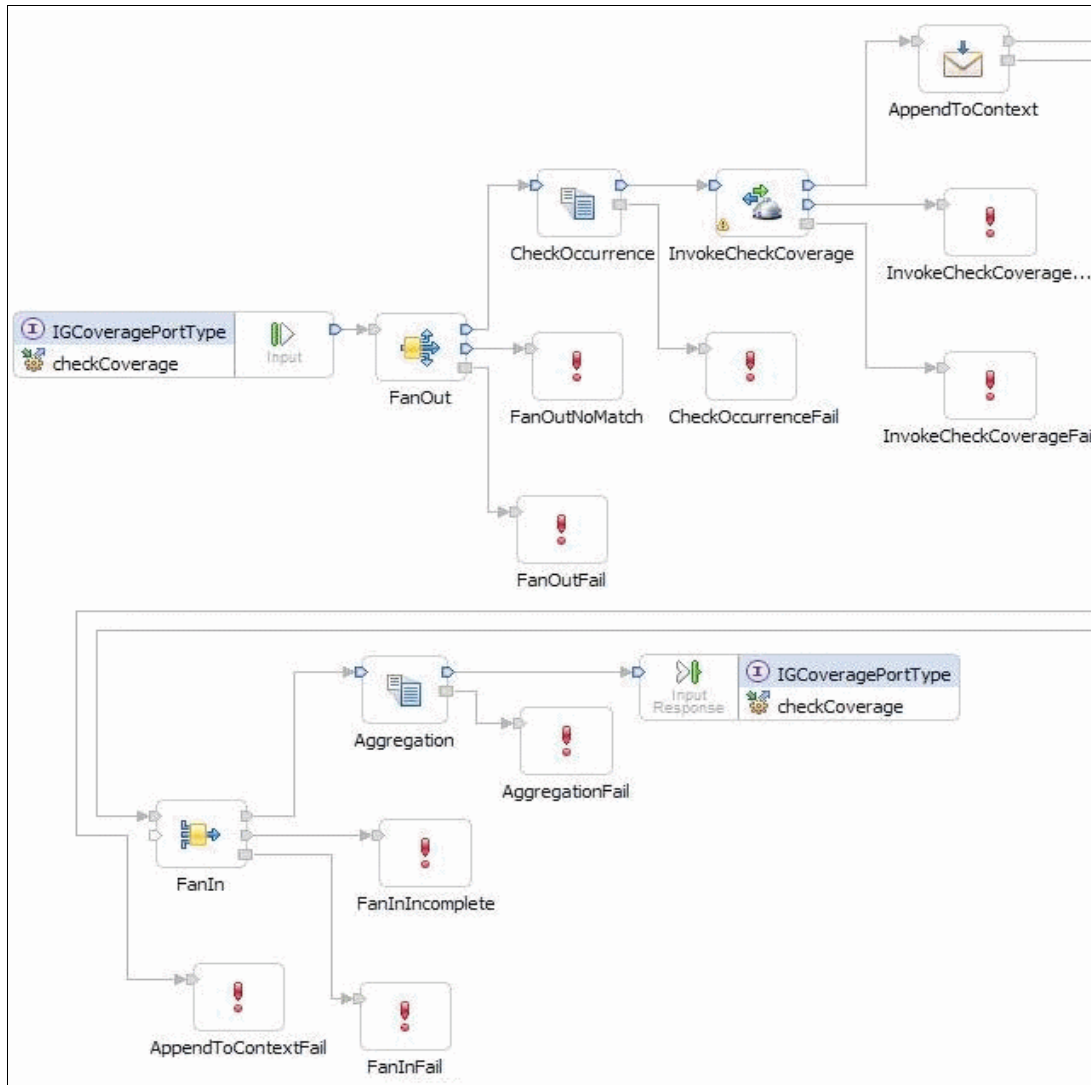


Figure 3-16 Aggregation Mediation sample

This mediation uses the FanIn, FanOut, XSLT Transform, and Message Element Setter primitives and the shared and primitive contexts to aggregate the

message flow. In this sample, the incoming message contains an array of query elements, and the aggregation functionality invokes a given service for each element in turn.

This type of processing might be used to process a car insurance quotation, invoking a given service to request costing information pertaining to each additional driver listed in the form.

Other factors

In this section, we describe other important factors that impact the estimation for sizing a solution.

Adapters

The use of adapters can affect workload, depending on how they are used (synchronous or asynchronous) and for what they are used (inbound or outbound). The following adapters can be used for an integration solution:

- ▶ SAP
- ▶ PeopleSoft
- ▶ Flat File
- ▶ JDBC
- ▶ Enterprise Content Management
- ▶ JD Edwards EnterpriseOne
- ▶ IBM Lotus® Domino®
- ▶ Oracle E-Business Suite
- ▶ Siebel
- ▶ IBM iSeries®

The desired topology

Low or high workload should be the starting point. The entire WebSphere ESB Registry Edition topology can be together in the same server or can be separated (as shown in Figure on page 70).

Expected peak throughput

Throughput refers to the roundtrip scenario execution, which is a key factor that indicates workload and directly affects the size of the solution. Throughput is important when defining nonfunctional requirements so that you can understand the number of requests to WebSphere ESB Registry Edition and to WSRR, and the possible estimated growth of these requests.

Expected CPU utilization

Expected CPU utilization refers to the maximum percentage of utilization of the CPU compared to the maximum workload. The normal CPU usage should be

closer to 0% than to 100%, but when the highest workload is reached (because of the peak throughput), the maximum desired CPU utilization is reached also.

3.4.2 Sizing WSRR

The workload in WSRR is mainly produced by lookups. However, some other factors can affect workload. We discuss those factors in this section.

Stored artifacts

The artifacts that are stored in the registry can include various types, from WSDL files and XSD files to complete enterprise archive files. Many of these artifacts result in heavy queries when looking for information about them. If the type of artifact is a WSDL file, the complexity of the artifact can affect workload. The deeper the hierarchy in the XML file, the higher the cost when parsing it. However, if these artifacts are generic documents, size is important. Larger files result in more workload.

The number of artifacts that are stored also impacts workload. However, there is not a large number of documents expected to be stored in the registry. Thus, even when indexes increase storage size, workload does not increase much.

Governance

If you use the governance enablement profile or another governance profile, validation and parsing is required. Therefore, workload is generated on every interaction with the registry. However, this workload is insignificant unless there are a large number of lookups to process.

Security

When security is enabled in the registry, it has an impact on workload. SSL-based solutions have overhead over plain channels and can increase workload. Depending on the level of security that is applied, the workload is different.

Connecting to other security service providers, for authorization, authentication, auditing, or whatever function must be used, can also increase workload from the WebSphere ESB Registry Edition side. For example, when consuming a web service, a SOAP message can be expensive in terms of the rules that are defined. If these rules are based on classifications, it can affect the workload.

User-defined concepts

Custom classifications, ontologies, relationships, and properties are not optimized for being part of the registry. The complexity and the amount of these user-defined concepts can make an impact on the workload.

Lookups and other queries

Lookups increase the workload in the registry. Queries are considered lookups and can also include basic database functions, such as creating, updating, and deleting artifacts in the registry. It is important to analyze all queries in the registry, but the most representative are those that are related to the dynamic lookup of web services from WebSphere ESB or other ESBs. Concurrency of these queries affects the workload in the final database and in the registry because XML parsing needs to be performed.

3.5 Monitoring the WebSphere ESB Registry Edition architecture

According to the SOA Governance life cycle, when WebSphere ESB Registry Edition is running, we must monitor the services running in our solution. To achieve this step in the life cycle, we need to monitor the complete SOA infrastructure, including resource use (CPU, memory, I/O, and so forth) and the behavior of services, such as SLAs (performance, availability, security issues, and so forth).

In WebSphere ESB Registry Edition, both WebSphere ESB and WSRR provide important information. WebSphere ESB has the runtime activity between service consumers and service providers. This information needs to be monitored. WSRR has all the service metadata, including SLAs. This information is also used for monitoring.

Therefore, with the WebSphere ESB information (which informs you about what is really happening in the runtime environment) and with the registry information (which describes how services should be working and how consumers should be consuming services), you can monitor services, register statistics, and also raise alarms if needed (for example, if an SLA is no longer accomplished).

3.5.1 IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager (ITCAM for SOA) interacts in many different ways with WebSphere ESB Registry Edition. It can read events from the components and can send events to the WSRR to update service metadata. Figure 3-17 illustrates the integration of ITCAM for SOA and WebSphere ESB Registry Edition.

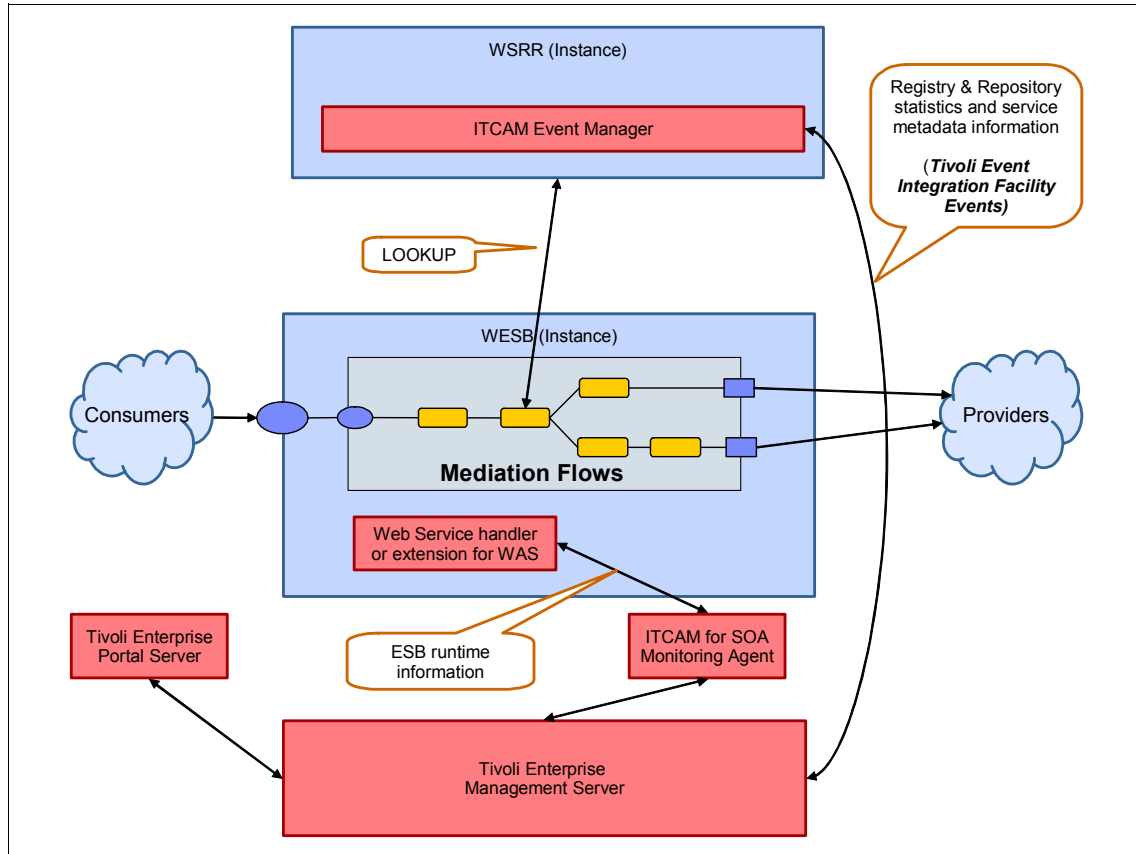


Figure 3-17 ITCAM for SOA and WebSphere ESB Registry Edition

WSRR settings

The ITCAM for SOA Event Handler for WSRR, shown in Figure 3-17 on page 74, runs in the WSRR instance that resides between WSRR and IBM Tivoli Monitoring. Those events that are detected by Tivoli Monitoring (for example, a new runtime service interface is down and no longer accessible) are used to send another event to the Event Handler (optionally using IBM Tivoli Enterprise Console® or Netcool/OMNIBus). The registry metadata is updated and the Event

Handler then creates, updates, or removes properties on the WSDLPort or SCA Export logical objects in the registry.

The ITCAM for SOA Event Handler is integrated with WSRR V7.5 (a Support Pack is required to add this capability for WSRR versions prior to V7.5). You can configure which objects (such as ServiceEndpoint, SOAPServiceEndpoint, and SCAServiceEndpoint) the metadata is stored on, and support is provided for both consumer and provider events.

The ITCAM Integration Editor is shown in Figure 3-18.

Event ID	Target Type	WSRR Property Name	Compound	Received Type	Received Value	Cleared Type	Cleared Value	
EventID1	Default	WSRRPropertyName1	<input checked="" type="checkbox"/>	Property Value	msg	Property Value	msg	Delete
EventID2	Default	WSRRPropertyName2	<input checked="" type="checkbox"/>	Static String	asdfasdf	Removal		Delete
EventID3	Default	WSRRPropertyName3	<input type="checkbox"/>	Static String	asdfasdf	Static String	cleared value	Delete
Fault_610	Default	Fault_610	<input type="checkbox"/>	Property Value	fault_count	Static String	cleared value	Delete
MaxMessageSize_6	Default	avg_msg_length_MMS_610	<input checked="" type="checkbox"/>	Property Value	avg_msg_length	Static String	cleared value	Delete
MaxResponseTimeC	Default	max_elapsed_time_MRTC_61	<input type="checkbox"/>	Property Value	max_elapsed_time	Static String	cleared value	Delete
MaxResponseTimev	Default	max_elapsed_time_MRTW_6	<input type="checkbox"/>	Property Value	max_elapsed_time	Static String	cleared value	Delete
MessageSize_610	Default	avg_msg_length	<input checked="" type="checkbox"/>	Property Value	avg_msg_length	Removal		Delete
ResponseTimeCritic	SOAPServiceEndp	max_elapsed_time_RTC_61C	<input type="checkbox"/>	Property Value	max_elapsed_time	Property Value	situation_name	Delete
ResponseTimeWarr	Default	avg_elapsed_time_RTW_61C	<input checked="" type="checkbox"/>	Static String	peak value	Static String	back to normal	Delete

Figure 3-18 ITCAM Integration Editor

You can find details about ITCAM for SOA and the ITCAM family of products in *IBM Tivoli Composite Application Manager Family Installation, Configuration, and Basic Usage*, SG24-7151.

Sample events to be monitored

Many SLAs can be defined for a service regarding performance, availability, security, and so forth. Example 3-1 provides one sample an SLA definition.

Example 3-1 Sample SLA definition

The consumer “Supply Company” will have an average response time of less than 1.5 second as long as there are no more than 20 request per minute, and the concurrency is not bigger than 2 requests at a time.

For this SLA (defined in WSRR), we must monitor the invocation to the service in WebSphere ESB. While collecting this information, ITCAM for SOA is processing (for example, every 5 minutes) if an issue occurs as described in the following examples:

- ▶ The time response of the service took more than 1.5 seconds.
- ▶ WebSphere ESB is receiving more than 20 request per minute.
- ▶ Three or more requests at the same moment were processed.

When an issue occurs that violates the SLA, ITCAM for SOA raises an alert. However, the SLA can change. Thus, ITCAM for SOA must also check WSRR to get the SLA information and then determine whether to raise an alarm. This alert system must be designed in the ITCAM for SOA monitoring model. Other events might generate changes in WSRR, and these changes are also triggered by ITCAM for SOA.



Implementing a Low Workload topology

This chapter provides detailed information about how to install and deploy WebSphere ESB Registry Edition in a simple topology, including both runtime components and development and configuration tools. We have termed this a Low Workload topology because it is not intended for significant production use.

This chapter also helps you to understand the components that are involved in the topology, and the basic dependencies between the components.

4.1 Topology description

As explained in Chapter 3, “Topology overview” on page 47, the Low Workload topology is designed for development environments or functional testing environments, such as a desktop development environment, an integration testing environment, or an acceptance testing environment. In this chapter, we implement a working environment that can serve as a starting point for becoming familiar with all of the components that comprise WebSphere ESB Registry Edition.

Figure 4-1 shows the main components that we install and configure in this chapter.

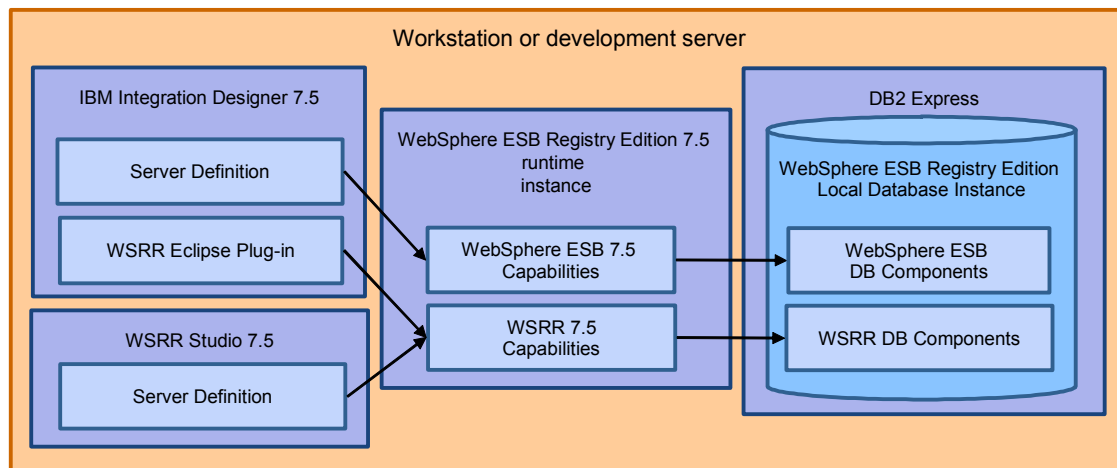


Figure 4-1 WebSphere ESB Registry Edition for development

4.2 Topology implementation road map

We base the implementation of the Low Workload topology in a “step-by-step” road map, as shown in Figure 4-2.

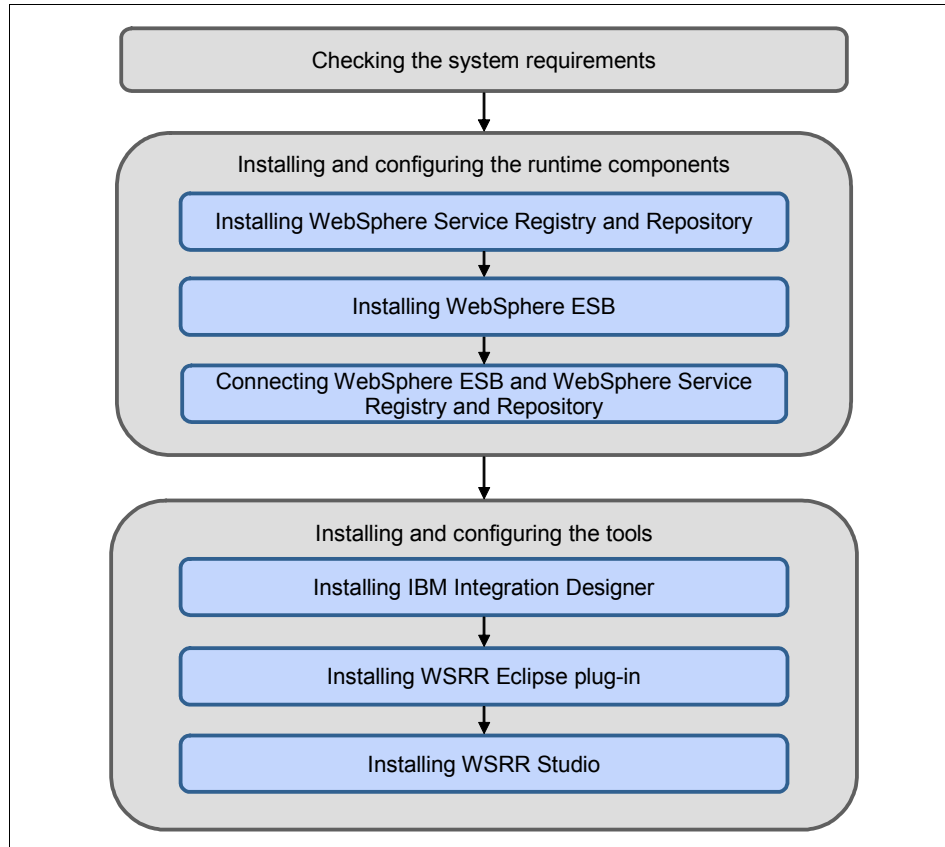


Figure 4-2 WebSphere ESB Registry Edition deployment road map

For the purposes of this book, we install and configure all the components that are provided with WebSphere ESB Registry Edition, including the runtime components and the tools, so that at the end of this chapter we have a running stand-alone development environment. For other purposes (for example, to deploy a “shared” integration test environment), we can skip the step to install and configure the tools.

Although we use this road map for our implementation, other methods of installing and configuring the environment can achieve similar results.

4.3 Checking system requirements

Before you install and configure WebSphere ESB Registry Edition, verify that your target platform meets the hardware and software system requirements. For example, consider the following important characteristics:

- ▶ Hardware
 - Check processor architecture
 - Check available memory
 - Check disk space
- ▶ Software
 - Check operating system and required fix packs
 - Be sure to use credentials with enough privileges
 - Check web browser version
 - Check database requirements

For the purposes of this book, we used a platform with the following characteristics:

Processor	Intel Core 2 Duo, at 3 GHz
RAM memory	3.00 GB
Disk space	50.00 GB
Operating system	Microsoft Windows Server 2003 Standard Edition
Installation user	As administrator
Web browser	Mozilla Firefox 3.6
Database	DB2 Express Edition

DB2 Express Edition note: DB2 Express Edition is now bundled with WebSphere ESB Registry Edition. It is the natural replacement for Cloudscape (Derby) for those scenarios where an enterprise-level database is not required, such as development environments or “pilot” projects.

For detailed information about system requirements, refer to the following links:

- ▶ WebSphere Enterprise Service Bus system requirements
<http://www.ibm.com/support/docview.wss?uid=swg27020614>
- ▶ WebSphere Service Registry and Repository system requirements
<http://www.ibm.com/software/integration/wsrr/sysreqs/#wsrrv75>
- ▶ IBM Integration Designer system requirements
<http://www.ibm.com/software/integration/integration-designer/sysreqs/>

4.4 Installing and configuring the runtime components

In this section, we explain how to install and configure the runtime components that are needed to create a stand-alone WebSphere ESB Registry Edition instance, as shown in Figure 4-3.

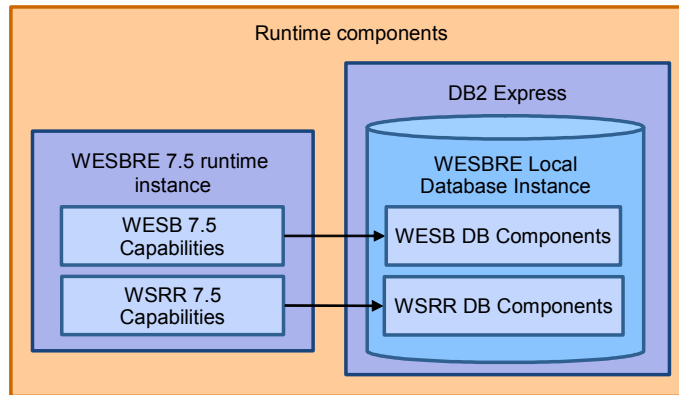


Figure 4-3 WebSphere ESB Registry Edition runtime components

In our scenario, we have one WebSphere Application Server instance serving the role of both the service registry and the enterprise service bus at the same time. For a development environment, this configuration is a useful alternative, because it requires less system resources than two separate application server instances. However, in this configuration, starting the application server can take more time. We use also one database instance for the configuration of both WebSphere Enterprise Service Bus (WebSphere ESB) and WebSphere Service Registry and Repository (WSRR).

4.4.1 Installing and configuring WSRR

In this section we show how to install WSRR and other required packages, such as DB2 Express Edition and IBM Installation Manager. The WebSphere ESB Registry Edition distribution includes these packages and allows you to install everything in one step using the Launchpad tool. The installation process that we use here also creates a runtime application server instance that supports the service registry capabilities. Later in this chapter, we augment this instance to also support the enterprise service bus capabilities. You can find additional information about how to install WSRR under various conditions at:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/twsr_welcome_wsrr_ins.html

Use the Launchpad tool as follows:

1. Locate and run the launchpad.exe file (or the corresponding file if your platform is not based on the Windows operating system) for the WSRR component.

64-bit Windows: On 64-bit Windows operating systems you must right click **Launchpad.exe** and select **Run as Administrator** so the install will be able to create user accounts.

2. The Launchpad tool allows you to select between a typical installation or a custom installation, as shown in Figure 4-4. Choose **Install using typical installer**.

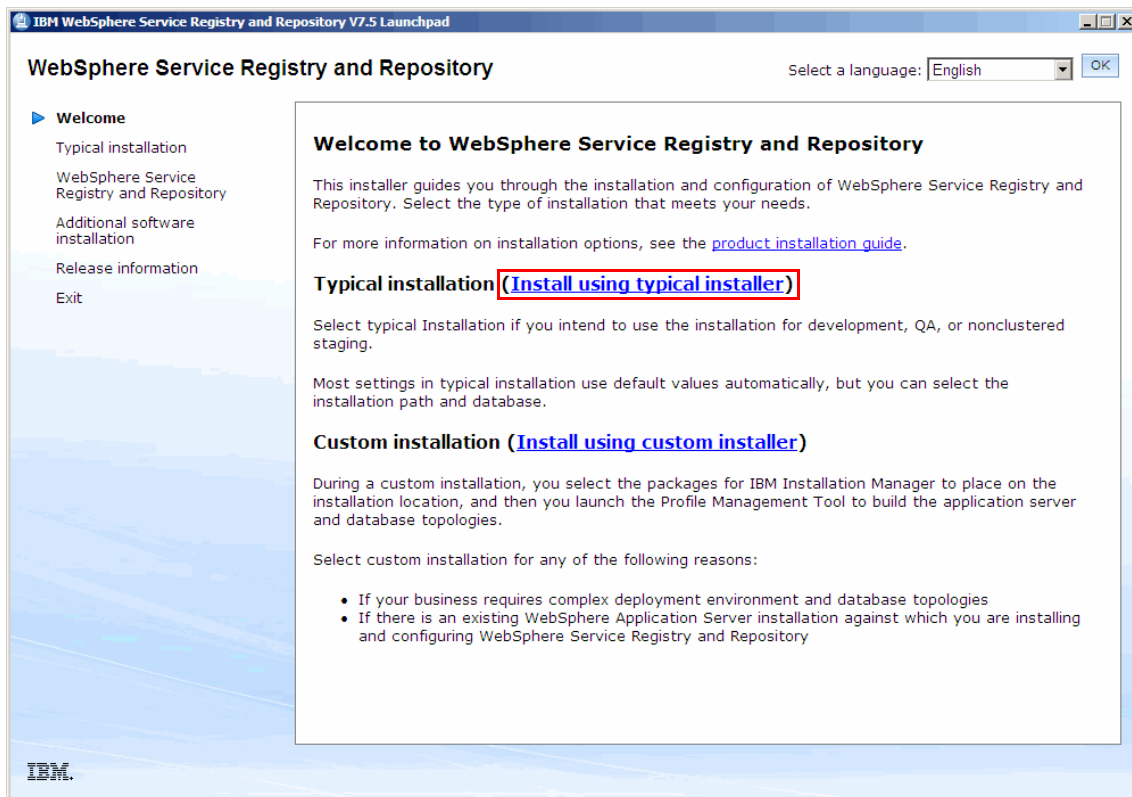


Figure 4-4 Launchpad tool welcome panel

3. Review the default values for the available options, as shown in Figure 4-5, and change these values if needed. Click **Next**.

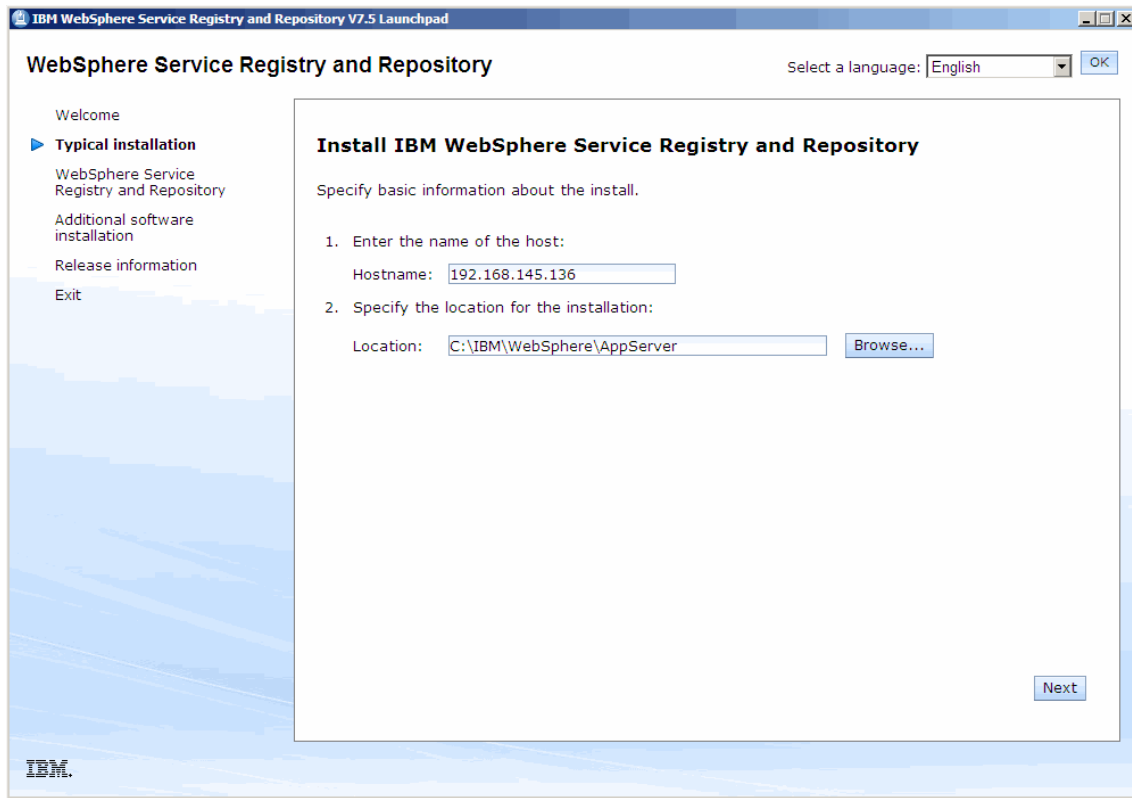


Figure 4-5 WSRR typical installation: Basic information panel

4. In the next step, provide information for the database configuration. Select the “Install an embedded DB2 Express database” option as shown in Figure 4-6, and click **Next**.

DB2 Express Edition note: Do not install DB2 Express Edition if the target system already contains a DB2 installation. In this case, check that the existing DB2 installation meets the system requirements, and select the “Use an existing database” option instead. Refer to the information center product documentation for specific instructions about how to install WSRR on an existing database.

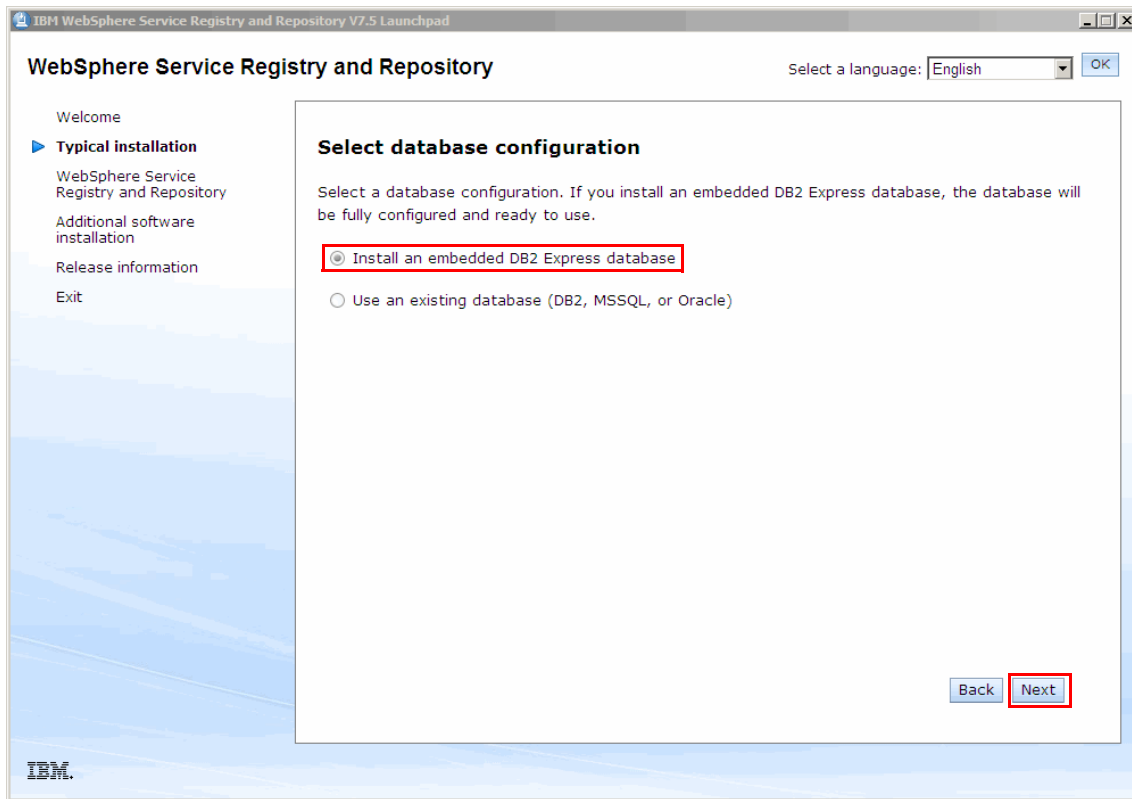


Figure 4-6 WSRR typical installation: Database configuration panel

5. Review the installation summary (Figure 4-7), and accept the license agreement. Click **Install Software** to launch the installation and configuration task.

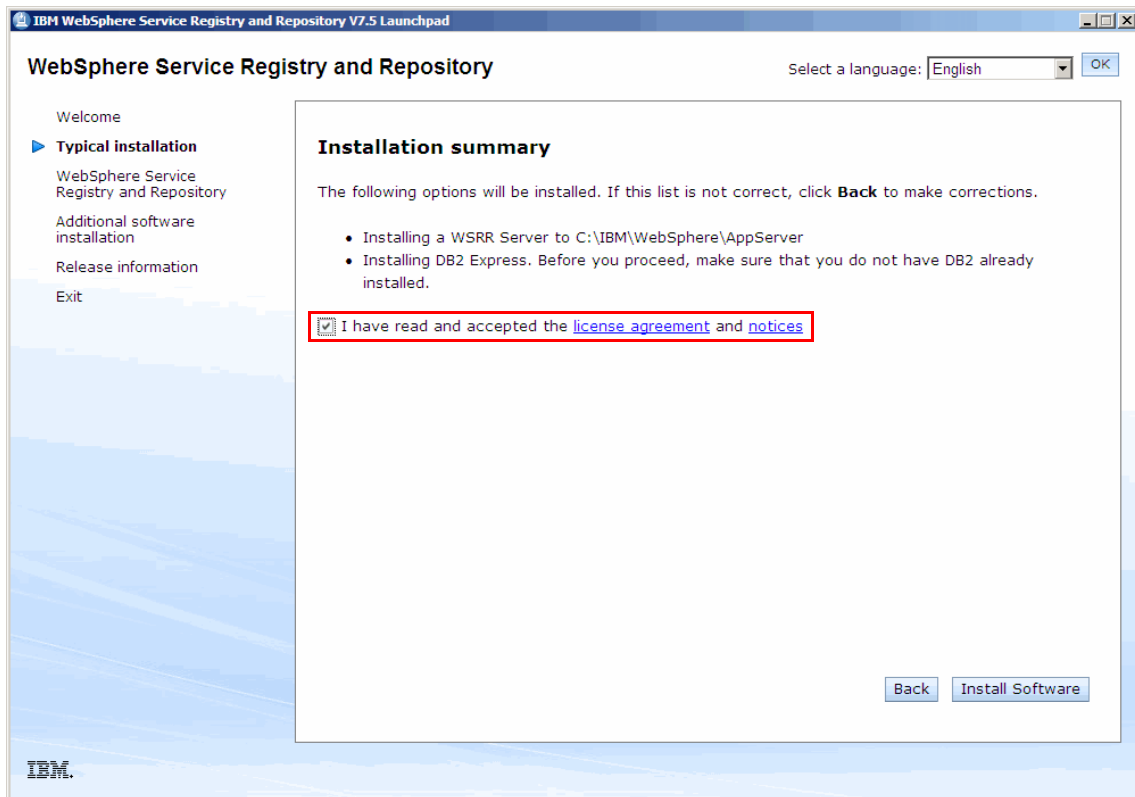


Figure 4-7 WSRR typical installation: Installation summary form

6. The installation process can take from 20 to 60 minutes, depending on the performance of the target system. While the installation completes, you can print the installation information as shown in Figure 4-8. The progress bar shows the task status.

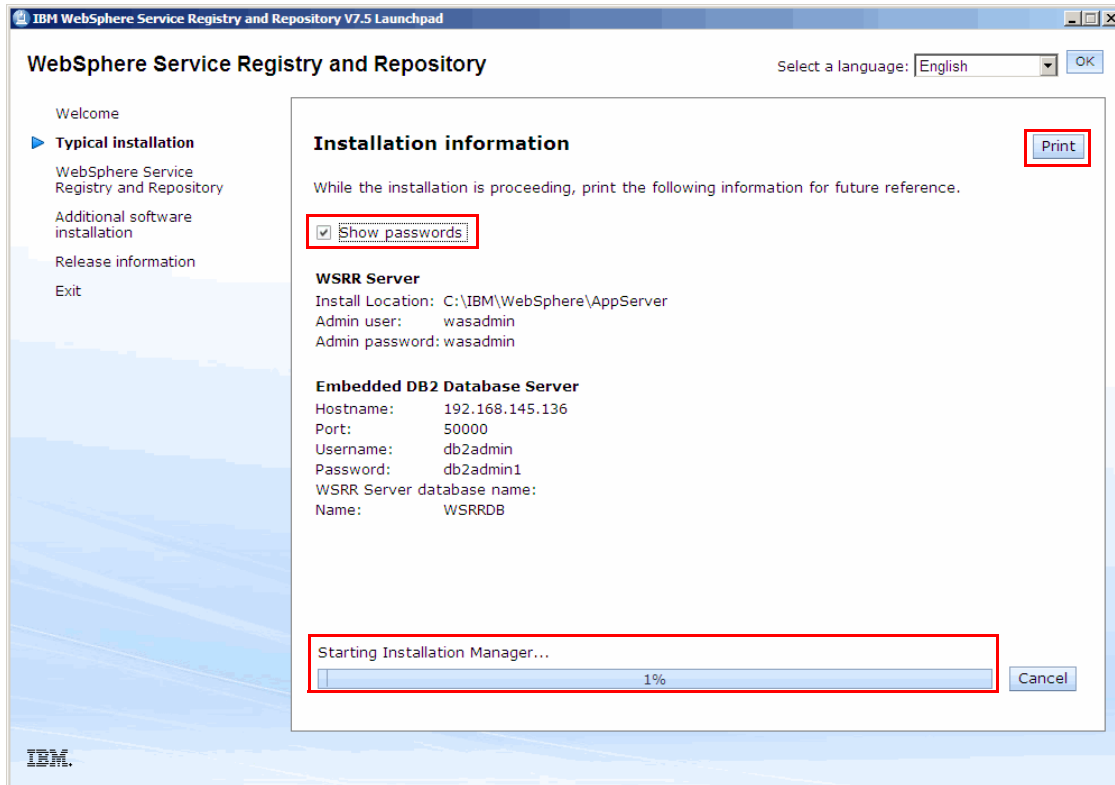


Figure 4-8 WSRR typical installation: Installation information

7. After a successful installation, a message displays as shown in Figure 4-9. Click **No**, and then close the Launchpad window.

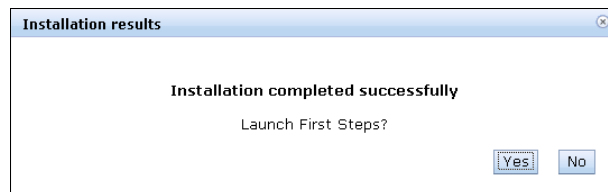


Figure 4-9 Installation complete

Now the target system contains the following components:

- A WebSphere Application Server installation, including support for WSRR and the Business Space capabilities
- A DB2 Express Edition installation

- A WebSphere Application Server stand-alone server profile, based on the WSRR template, including Business Space
 - A DB2 Express Edition database instance, containing the required components (tables, procedures, and so forth) for the stand-alone server profile
8. At this point, load and activate a configuration profile for WSRR. Load and activate the Governance Enablement Profile that is included with the product. You can find details about how to load and activate a profile at:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/twsr_configrn_config_profiles_working_with.html

For a discussion about the profile features, see Chapter 6, “Governance enablement profile” on page 143.

4.4.2 Installing and configuring WebSphere ESB

In this section, we explain how to install WebSphere ESB and apply its capabilities to the stand-alone server that we created in 4.4.1, “Installing and configuring WSRR” on page 81. This process consists of the following tasks:

1. Installing the WebSphere ESB binaries on the same package on which we installed WSRR
2. Augmenting the existing application server profile with the WebSphere ESB capabilities

You can find additional information about installing and configuring WebSphere ESB under various conditions at:

http://publib.boulder.ibm.com/infocenter/esbsoa/wesbv7r5/index.jsp?topic=/com.ibm.websphere.wesb.install.doc/topics/cins_inst_cnfg_bpm.html

Installing the WebSphere ESB binaries

In this section, we install the WebSphere ESB binaries on top of the existing installation of WSRR. We also use the Launchpad tool for this task. After this task, the installation is extended with new capabilities in the form of *profile templates*. We can use these profile templates to create new WebSphere ESB profiles or to augment existing profiles.

Use the Launchpad tool as follows:

1. Locate and run the `launchpad.exe` file (or the corresponding file if your platform is not based on the Windows operating system) for the WebSphere ESB component.
2. The Launchpad tool allows you to perform several tasks, as shown in Figure 4-10. Select **WebSphere ESB** from the top-left menu, and click **Install**.



Figure 4-10 WebSphere ESB: Launchpad tool

3. The installation wizard recognizes that several of the components that are needed for WebSphere ESB are already installed in the system. It asks whether you want to install the same components in another group. For our scenario, we want to install WebSphere ESB on the same installation group where WSRR is installed. Click **Cancel**, as shown in Figure 4-11, to deselect the previously installed packages.

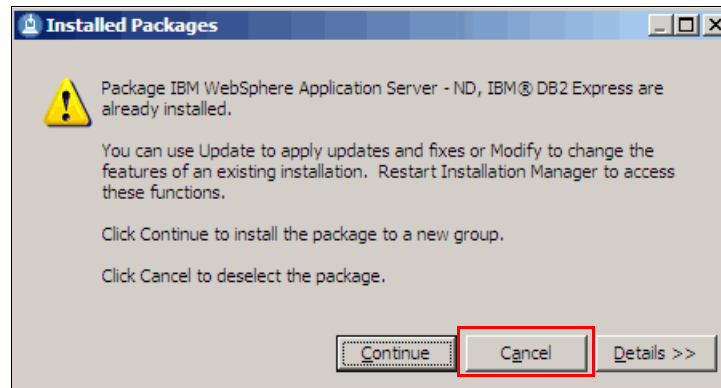


Figure 4-11 Previously installed packages dialog box

- The next panel shows all the packages that are included in the WebSphere ESB installation, and it allows you to select or clear manually the packages that you want to install. Verify that the selected packages are those shown in Figure 4-12, and click **Next**.

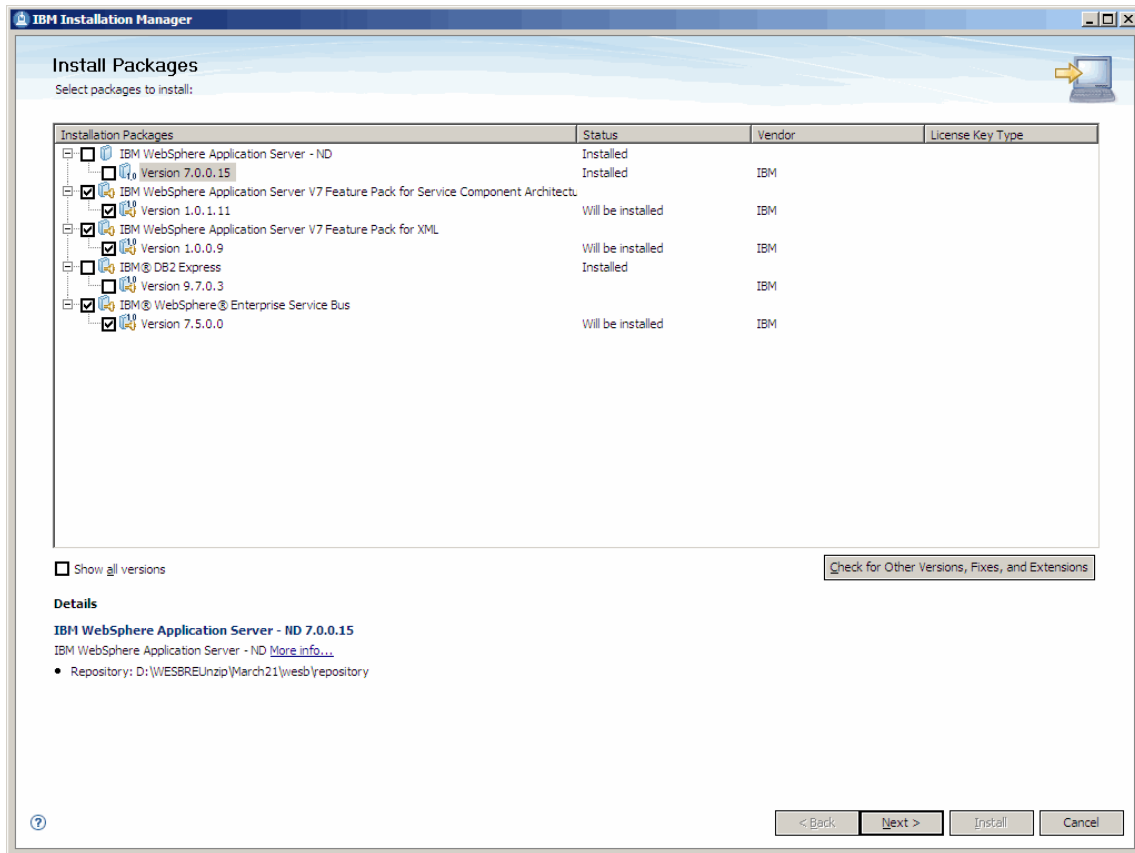


Figure 4-12 WebSphere ESB select packages to install panel

- In the next panel, read and accept the license agreement, and then click **Next**.

6. Verify that the selected package group is the same package where WSRR was installed (shown in Figure 4-13), and click **Next**.

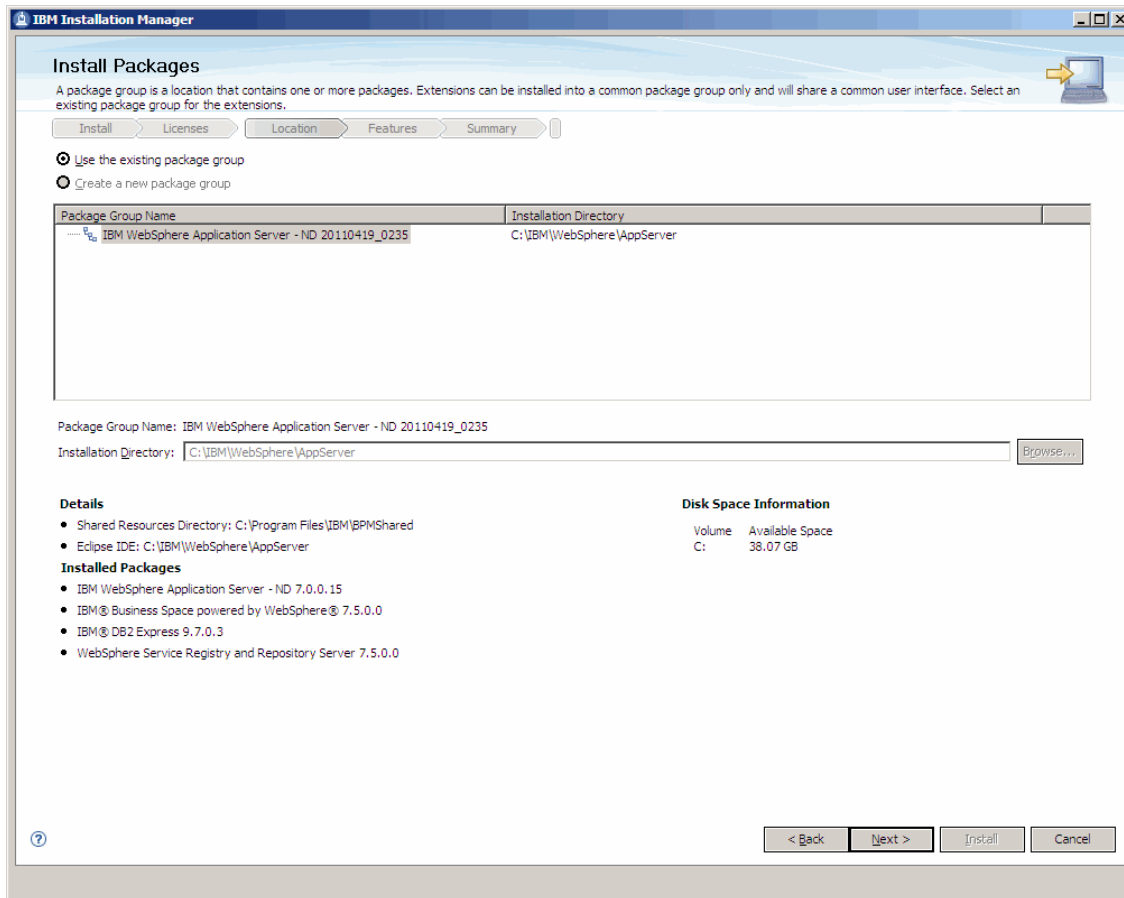


Figure 4-13 WebSphere ESB select package group panel

- Next, you can customize the installation by adding or removing elements for each main component. For the purposes of our scenario, we do not need to create a new application server for WebSphere ESB. Verify that the selected features are those shown in Figure 4-14, and click **Next**.

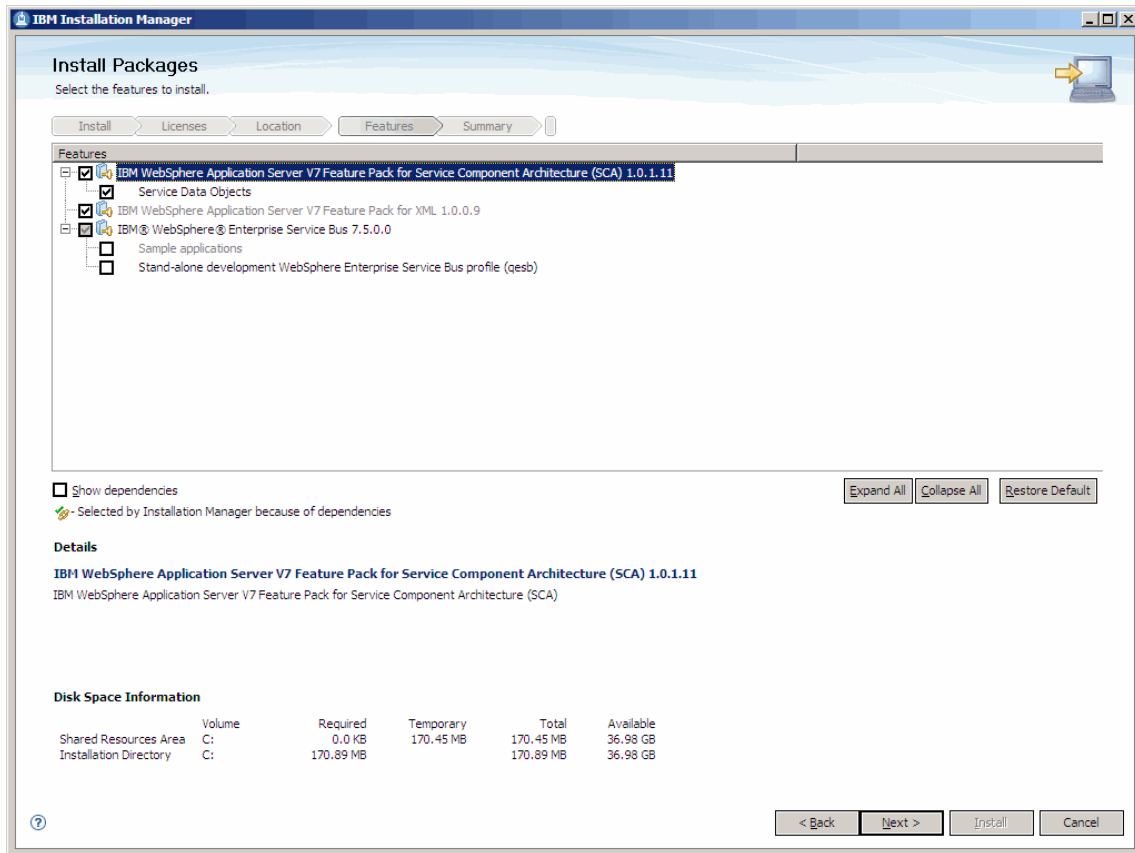


Figure 4-14 WebSphere ESB select features panel

- Finally, review the summary information and click **Install** to initiate the installation task. The installation process can take from 15 to 40 minutes, depending on the performance of the target system. A progress bar shows you the task status.

9. When the installation completes a panel displays a success message, as shown in Figure 4-15.
10. If the installation is successful, the environment can now create or add WebSphere ESB capabilities to new or existing application server profiles. Verify that the Profile Management Tool option is selected, and click **Finish** to proceed to the next task.

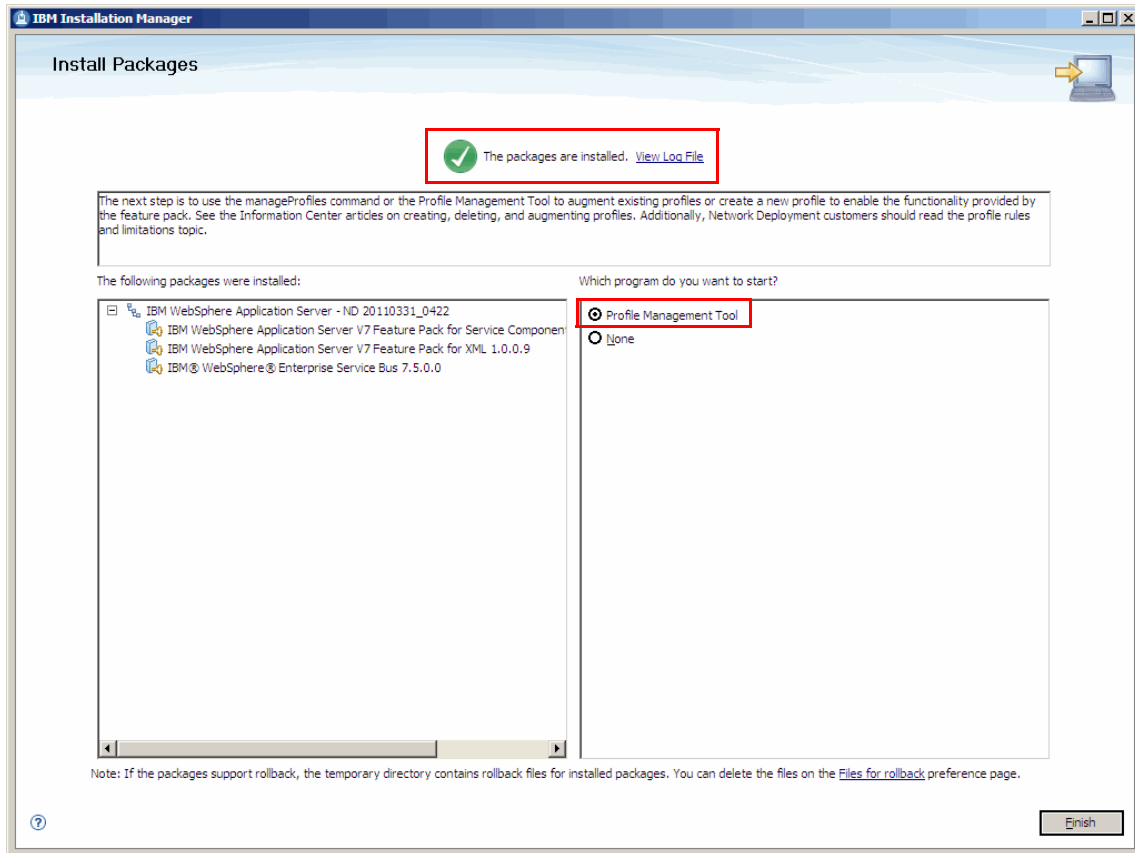


Figure 4-15 WebSphere ESB installation results panel

Augmenting the existing application server profile

In this section, we use the Profile Management Tool to modify our existing application server profile by augmenting it with the WebSphere ESB capabilities. Follow these steps to complete this task:

1. If the Profile Management Tool is not already running, locate and run the `pmt.bat` file in the `%WESBRE_ROOT%\bin\ProfileManagement` folder.
2. In the Welcome panel, click **Launch Profile Management Tool**. The Profiles tab lists the existing server profiles in the current installation package, as shown in Figure 4-16. Select the server profile that we created in the previous section, and click **Augment**.

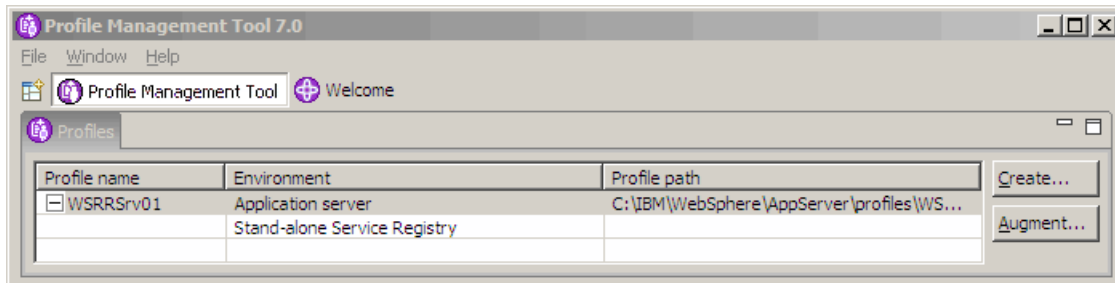


Figure 4-16 Profile Management Tool: Profiles tab

3. In the next panel, you can select augments that can be applied to the selected server profile. Select **Stand-alone enterprise service bus**, and click **Next**.
4. For profile augmentation options, select **Typical Profile Augmentation**, and click **Next**.
5. Provide the credentials for the profile's administrative user. The default values are those that were set when the profile was created, as shown in Figure 4-8 on page 86. Click **Next**.
6. For the database configuration, provide the options as shown in Figure 4-17. The database name is the same name that is used for the WSRR component, as shown in Figure 4-8 on page 86. Click **Next**.

Profile Management Tool 7.0

Database Configuration

Various components use WebSphere Enterprise Service Bus common database. Choose a database type and enter the information based on that type.

If the Profile Management Tool will run the database scripts as part of profile creation, do not specify an existing database in the Database name fields.

Select a database product:

DB2

☐ Override the destination directory for generated scripts.

Database script output directory:

Browse...

☐ Create a new local database.

☒ Use an existing local or remote database.

Common database name:

WSRRDB

☒ Run database scripts to create the database tables.

< Back Next > Finish Cancel

Figure 4-17 Profile Management Tool: Database configuration panel

7. The next panel shows additional database configuration options. Provide values as shown in Figure 4-18. Modify the database credentials according to the values that were set during the profile creation (see Figure 4-8 on page 86). Modify the database schema name as suggested, and then click **Next**.

Profile Management Tool 7.0

Database Configuration (Part 2)

Additional information is required to complete configuration for the DB2 database.

JDBC driver:

- ☒ DB2 Universal
- ☐ DB2 DataServer

User name to authenticate with the database:

db2admin

Password for database authentication:

Confirm password:

Location (directory) of JDBC driver classpath files:

\${WAS_INSTALL_ROOT}\db2\java

Browse...

Database server host name (for example IP address):

Win-SVR-2003-SE

Server port:

50000

Schema name:

WESB

< Back Next > Finish Cancel

Figure 4-18 Profile Management Tool: Database configuration panel (Part 2)

8. Finally, review the summary, and click **Augment** to initiate the augmentation task.
9. The process can take from 15 to 30 minutes, depending on the performance of the target system. After a successful completion, the Profile Augmentation Complete panel opens, as shown in Figure 4-19. Click **Finish**, and close the Profile Management Tool application.

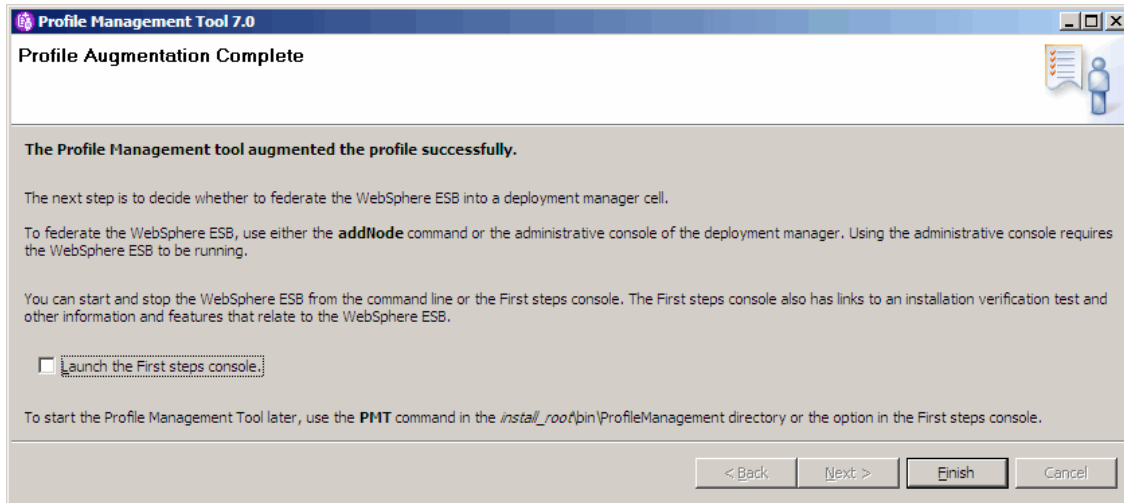


Figure 4-19 Profile Management Tool: Profile augmentation complete

4.4.3 Connecting WebSphere ESB and WSRR

This section explains how to configure the WebSphere ESB Registry Edition runtime components for scenarios in which a WebSphere ESB solution is supported by information that is stored and managed in a WSRR instance. Use the Integrated Solutions Console as follows:

1. Verify that the application server instance is running. Open a browser window to `http://<hostname>:9060/ibm/console`, and log in as administrator.
2. On the left side navigation tree, go to **Service Integration** → **WSRR Definitions**, as shown in Figure 4-20. Click **New** to register the server instance as a WSRR definition.

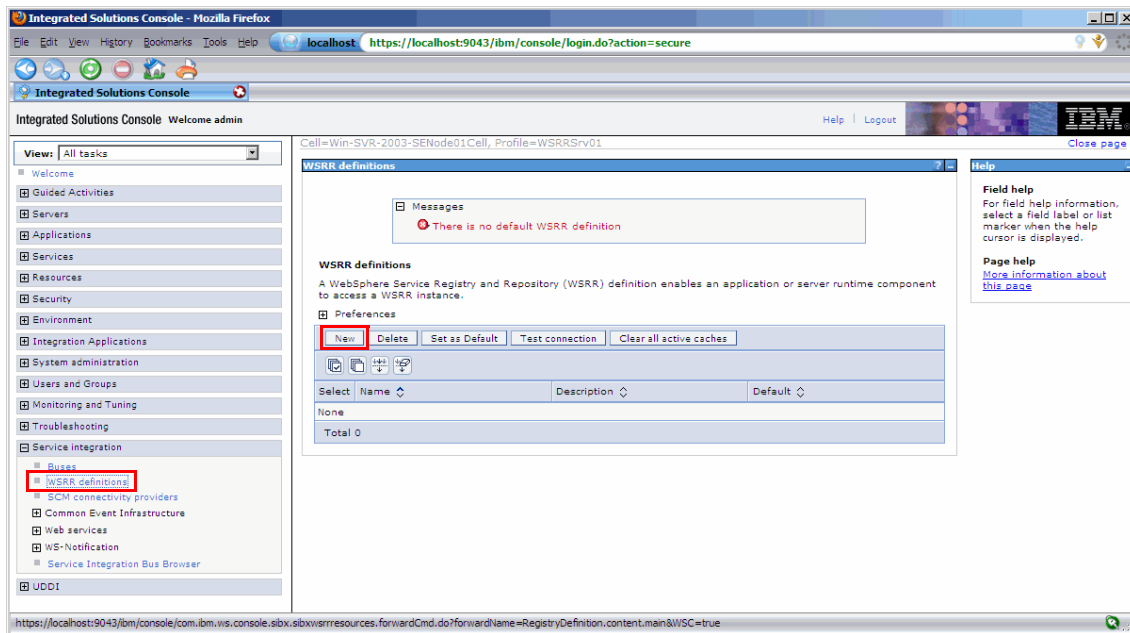


Figure 4-20 Integrated Solutions Console WSRR definitions

3. In the new WSRR definition form, enter a name and click **Apply**.
4. Then, click the **Connection properties** link, as shown in Figure 4-21.

[WSRR definitions](#) > WSRRDefault

A WebSphere Service Registry and Repository (WSRR) definition enables an application or server runtime component to access a WSRR instance.

Configuration

Test connection

General Properties	Additional Properties
<p>WSRR definition name</p> <input type="text" value="WSRRDefault"/>	<input checked="" type="checkbox"/> Connection properties
<p>Description</p> <div></div>	
<p>Default WSRR definition</p> <input type="text" value="Yes"/>	
<p>Timeout of cache</p> <input type="text" value="300"/> seconds	
<p>Connection type</p> <input type="text" value="Web service"/>	
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

Figure 4-21 Integrated Solutions Console: New WSRR definition

5. In the connection properties form, modify the field values, as shown in Figure 4-22. Use the **JAAS - J2C authentication data** link to create a new authentication alias before submitting the form. For now, use the WebSphere Application Server administrator credentials when creating the authentication alias. Click **OK**, and then **Save** to persist the changes for the server master configuration.

Figure 4-22 Integrated Solutions Console: Connection properties for WSRR definition

Runtime components deployed in separate profiles: In a topology where the runtime components are deployed in separate profiles, you must modify the SSL configuration so that the server certificate for the WSRR instance is imported into the signer certificates trust store of the WebSphere ESB instance. Refer to the following link for details:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.express.doc/info/exp/ae/usec_sslsignercerts.html

6. Finally, from the WSRR definitions panel, select the newly created entry, and click **Test Connection** to verify it, as shown in Figure 4-23.

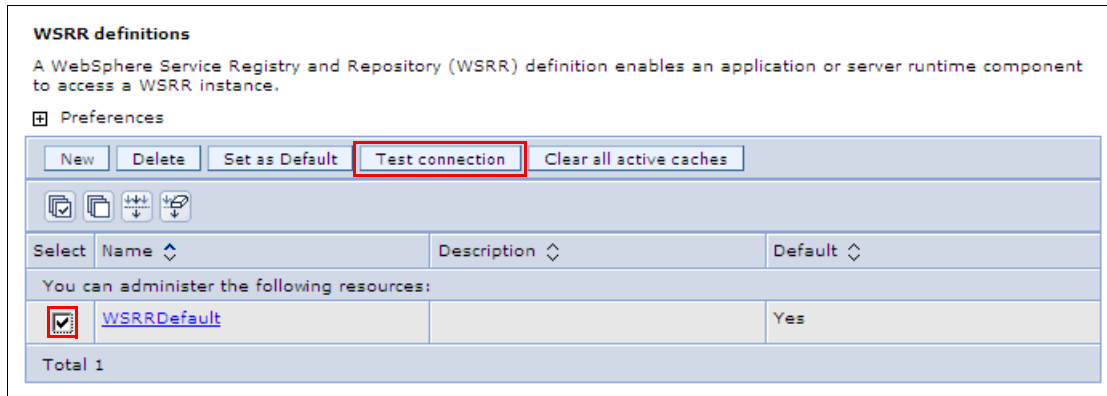


Figure 4-23 Integrated Solutions Console, test connection for WSRR definitions

4.5 Installing and configuring the tools

This section provides information about the basic installation of the tools that are included with WebSphere ESB Registry Edition. It also describes the configuration tasks that are needed to use the previously created runtime environment from the various tools. Figure 4-24 shows the tools that we discuss in this section.

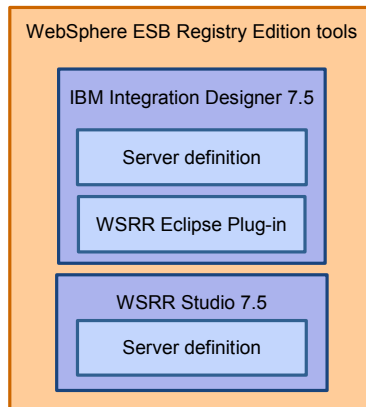


Figure 4-24 WebSphere ESB Registry Edition tools

4.5.1 Installing and configuring IBM Integration Designer

IBM Integration Designer is the development environment for WebSphere ESB. In this section, we describe the following tasks:

1. Installing the Integration Designer package
2. Configuring a connection to an existing server run time

You can find additional information about how to install Integration Designer at the following site:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=/com.ibm.wbpm.wid.imuc.doc/topics/c_inintro.html

Installing Integration Designer

Integration Designer includes its own version of the Launchpad tool, which we used in previous sections. To install this component, use the Launchpad tool as follows:

1. Locate and run the `launchpad.exe` file (or the corresponding file if your platform is not based on the Windows operating system) for the Integration Designer component.

The Launchpad tool allows you to perform several tasks, as shown in Figure 4-25. In our case, we have already installed a runtime environment that we will use as our test environment, so select **IBM Integration Designer** and click **Install Selected**.

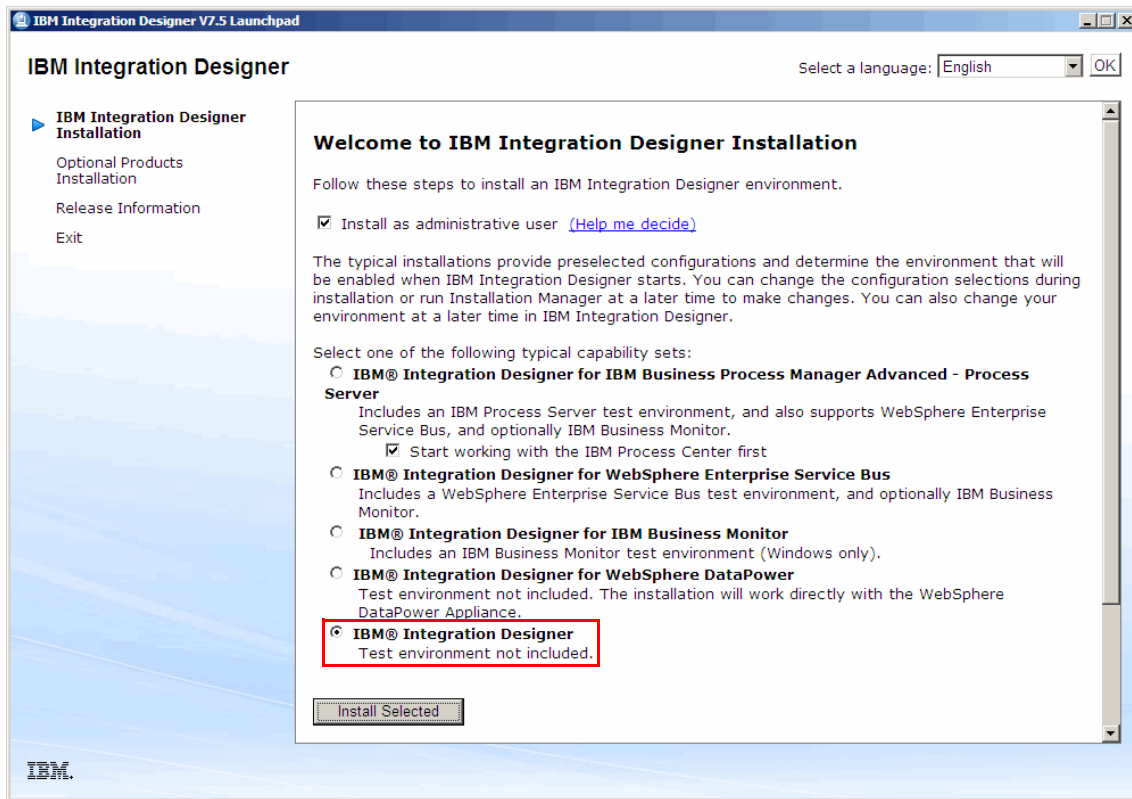


Figure 4-25 IBM Integration Designer Launchpad welcome panel

2. Next, select the packages shown in Figure 4-26, and click **Next**.

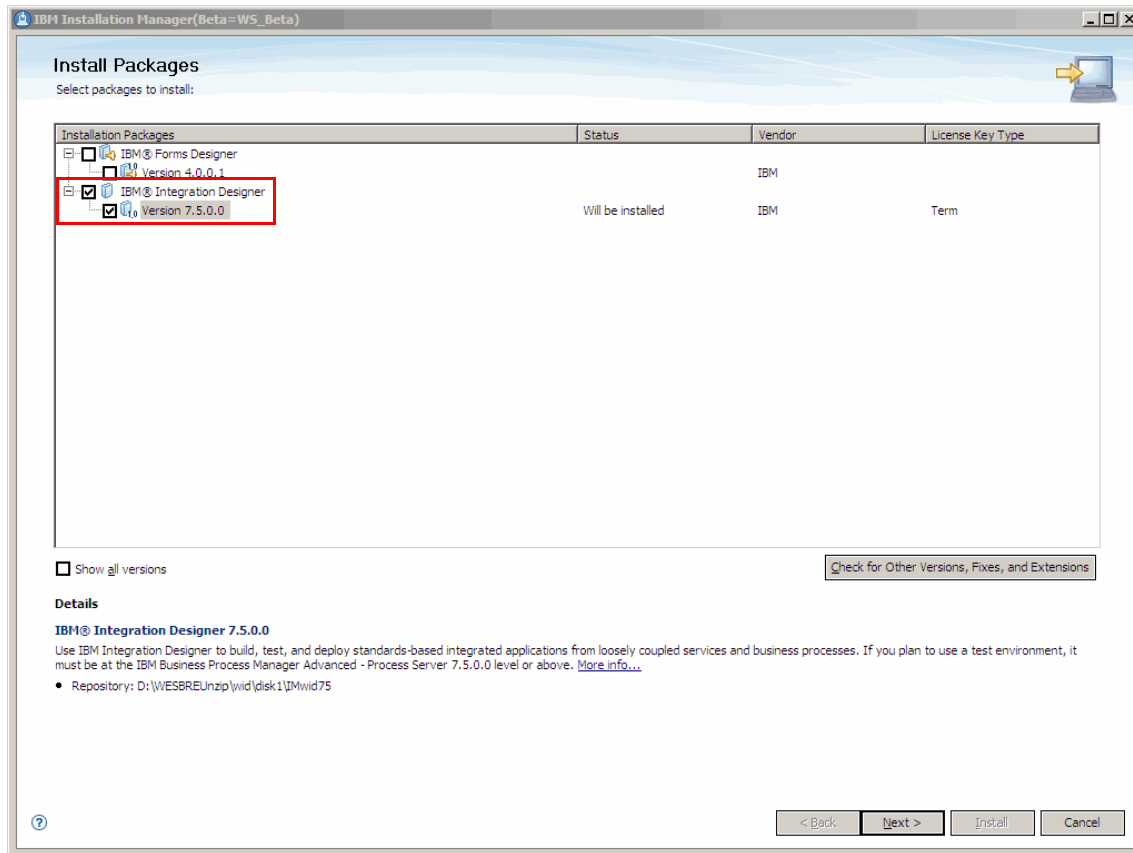


Figure 4-26 IBM Integration Designer: Package selection

3. In the next panel, read and accept the license agreement, and click **Next**.
4. Select **Create a new package group**. Review the package group installation directory, and modify the directory if needed. Click **Next**.

Installation note: You can also install Integration Designer into an existing package, such as Rational Software Architect or Rational Application Developer, if the versions of both the existing package and Integration Designer are compatible.

5. In the Translation panel, select the appropriate language support that you need to use the tool. The default language is English. Click **Next**.

- Determine the subcomponents that you want to include in this installation. Select the features as shown in Figure 4-27, and click **Next**.

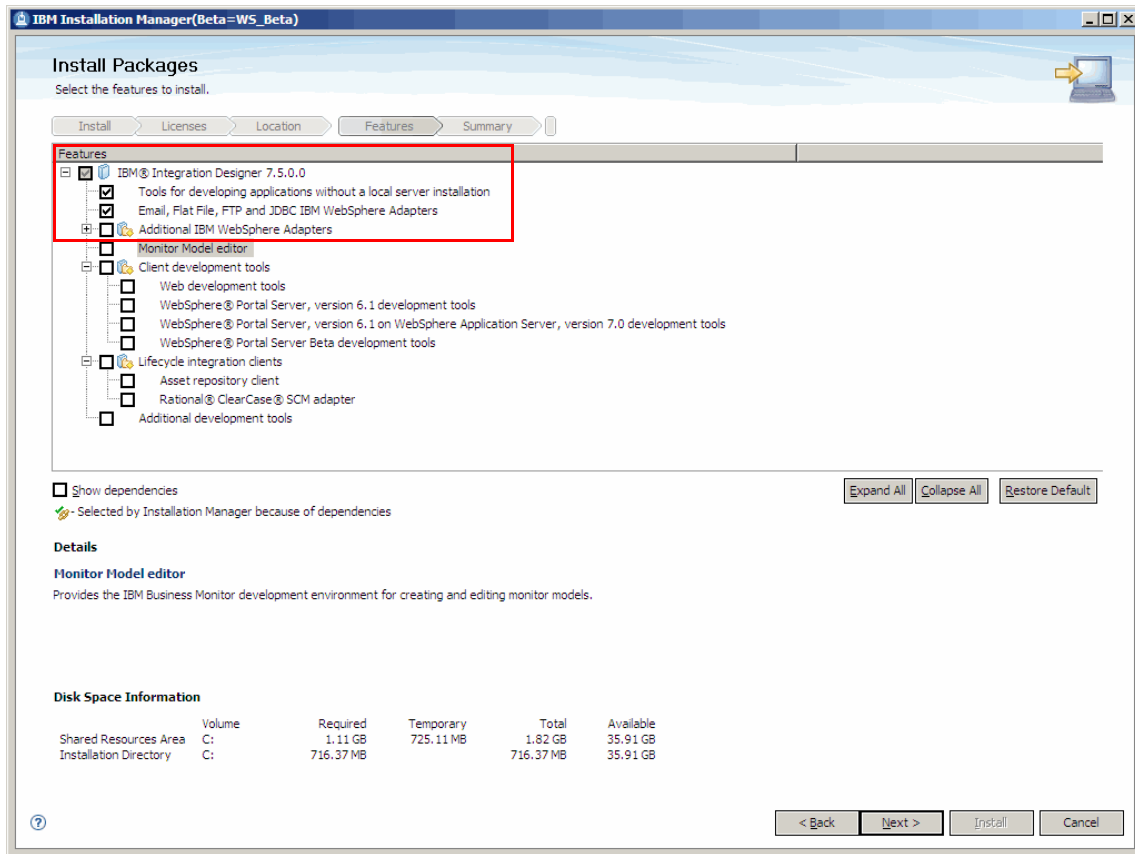


Figure 4-27 IBM Integration Designer select features

- Finally, review the summary information, and click **Install** to initiate the installation task.

8. The process takes from 15 to 40 minutes, depending on the performance of the target system. If the installation is successful, a message displays as shown in Figure 4-28. Click **Finish** to close the installation window.

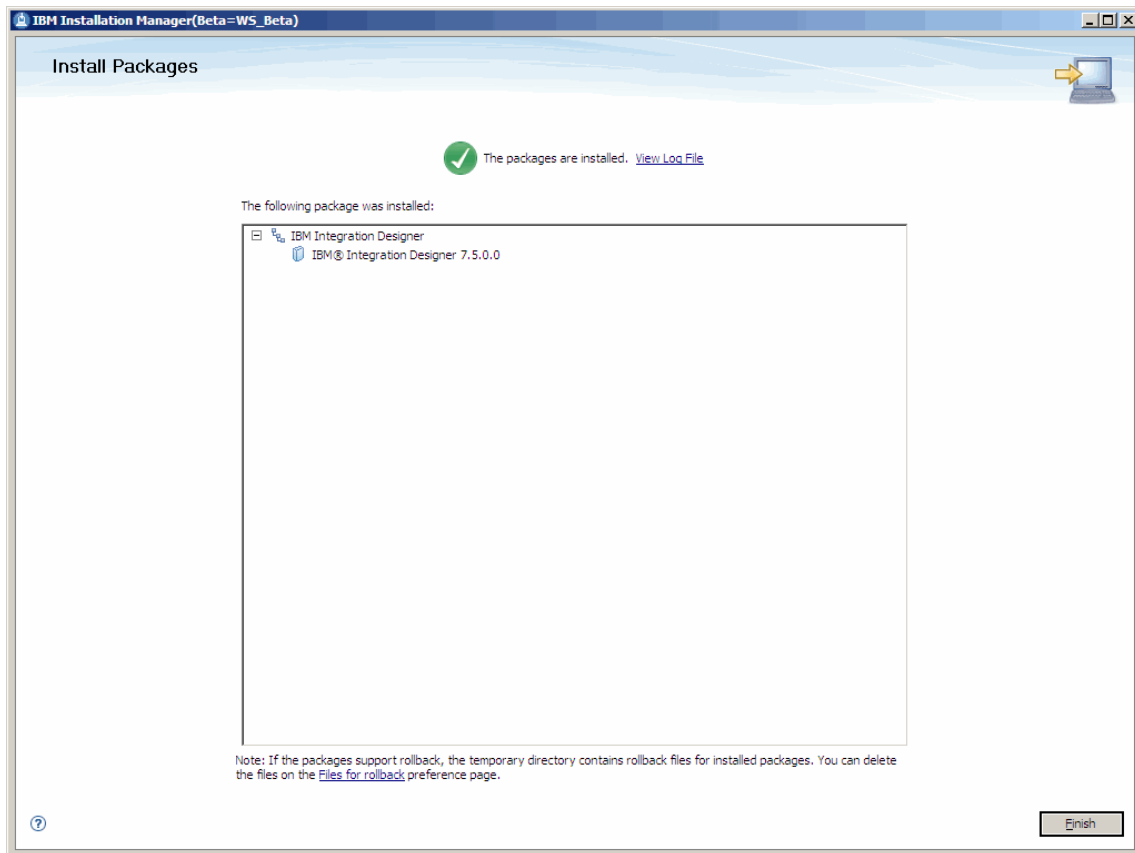


Figure 4-28 IBM Integration Designer installation results

Configuring a connection to an existing server run time

This section explains how to integrate the runtime environment installed in 4.4.2, “Installing and configuring WebSphere ESB” on page 87 as a test environment on Integration Designer.

Attention: If you switch to another workspace, you must repeat these steps to configure a connection to an existing environment. For our scenario, we use an existing stand-alone run time for the test environment that was not installed as a part of the Integration Designer installation.

Follow these steps:

1. Start Integration Designer. The Integration Designer workspace launcher prompts you for the location to create the workspace as shown in Figure 4-29. Provide a workspace location, and click **OK**.

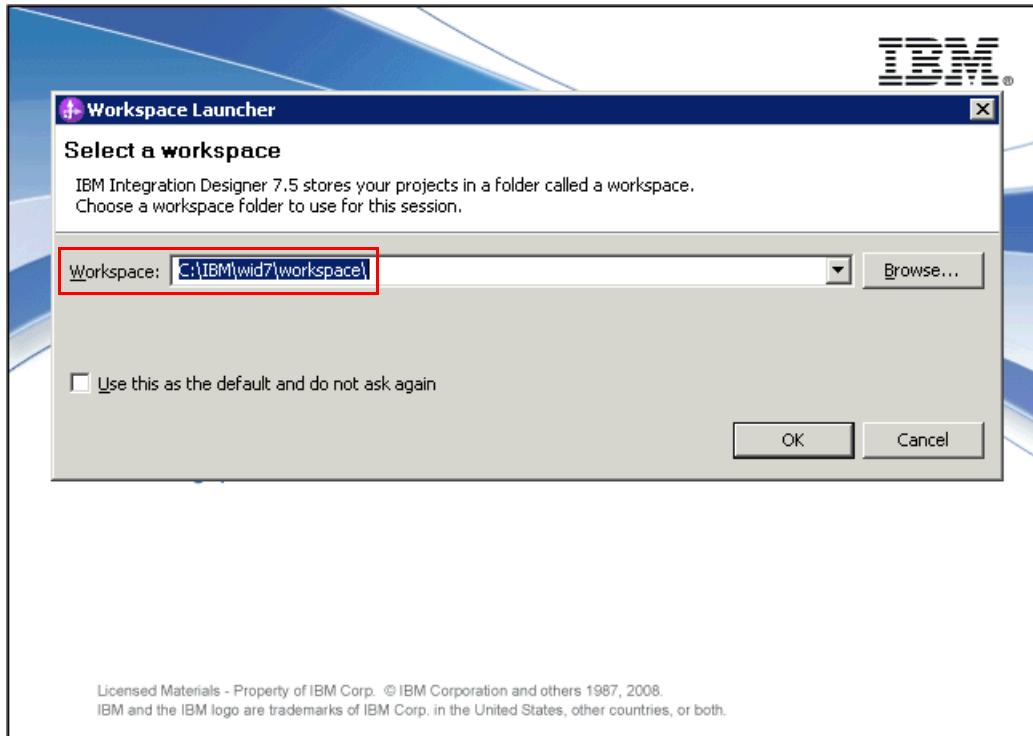


Figure 4-29 Integration Designer workspace launcher

2. Integration Designer opens in the Process Center perspective. Switch to the Business Integration perspective by selecting **Windows** → **Open Perspective** → **Business Integration** in the menu bar, as shown in Figure 4-30.

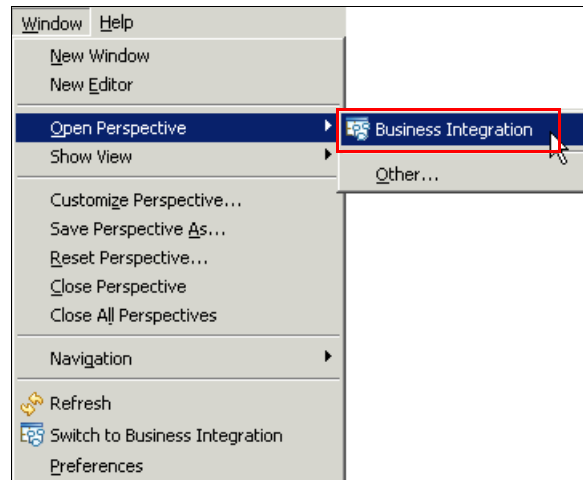


Figure 4-30 Open Business Integration perspective

After you successfully switch to the Business Integration perspective, Integration Designer looks as shown in Figure 4-31.

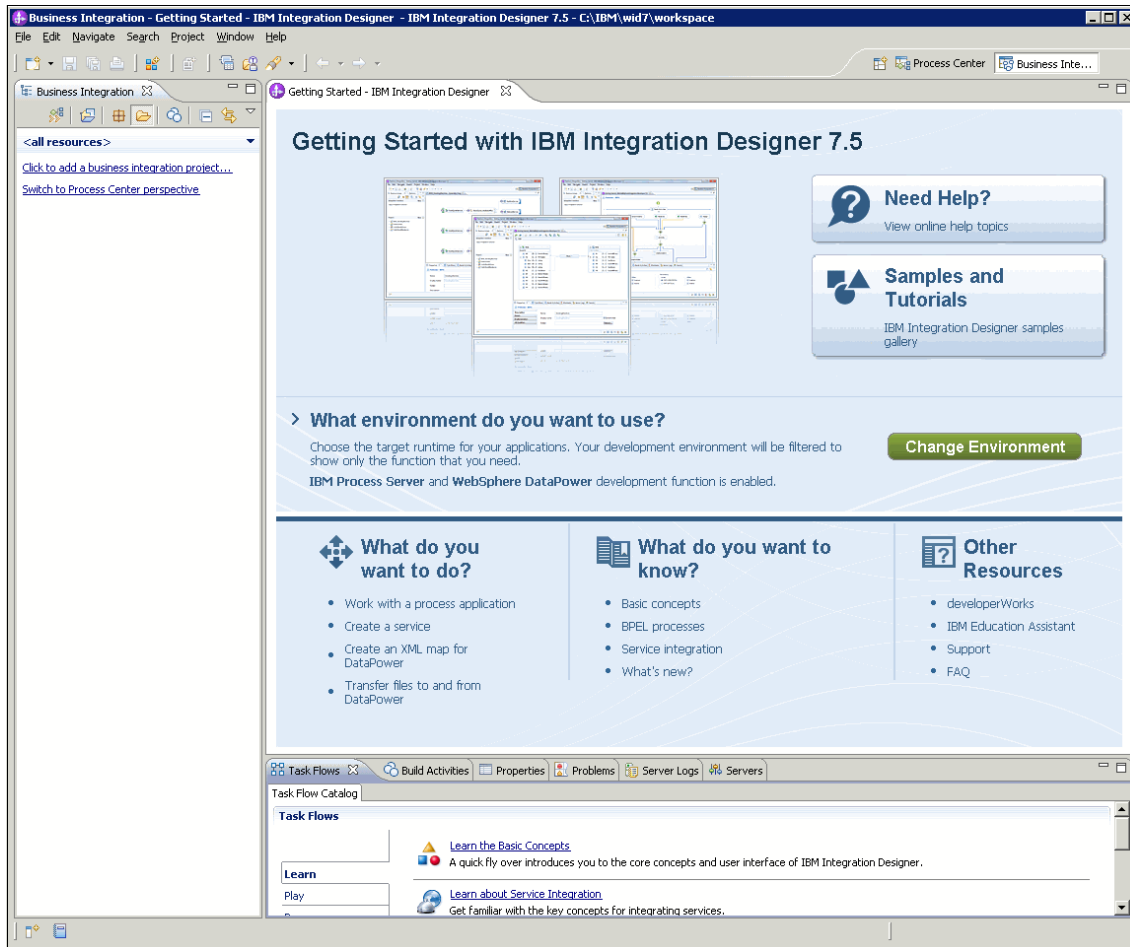


Figure 4-31 Getting Started with IBM Integration Designer

3. We now configure Integration Designer to interact with WebSphere ESB. On the Servers tab at the bottom of the window, right-click, and choose **New** → **Server** (see Figure 4-32).

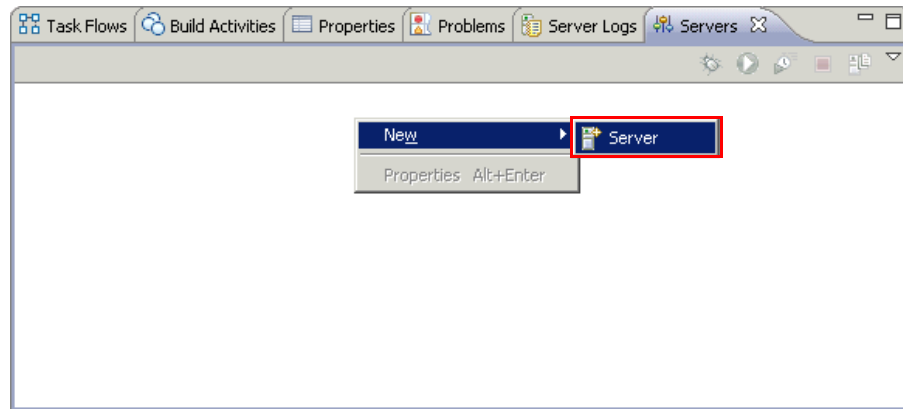


Figure 4-32 Creating a new test environment server

4. In the New Server window, select **WebSphere ESB Server 7.5**, as shown in Figure 4-33. Do not modify the remaining default values. Click **Next** to create the new runtime environment.

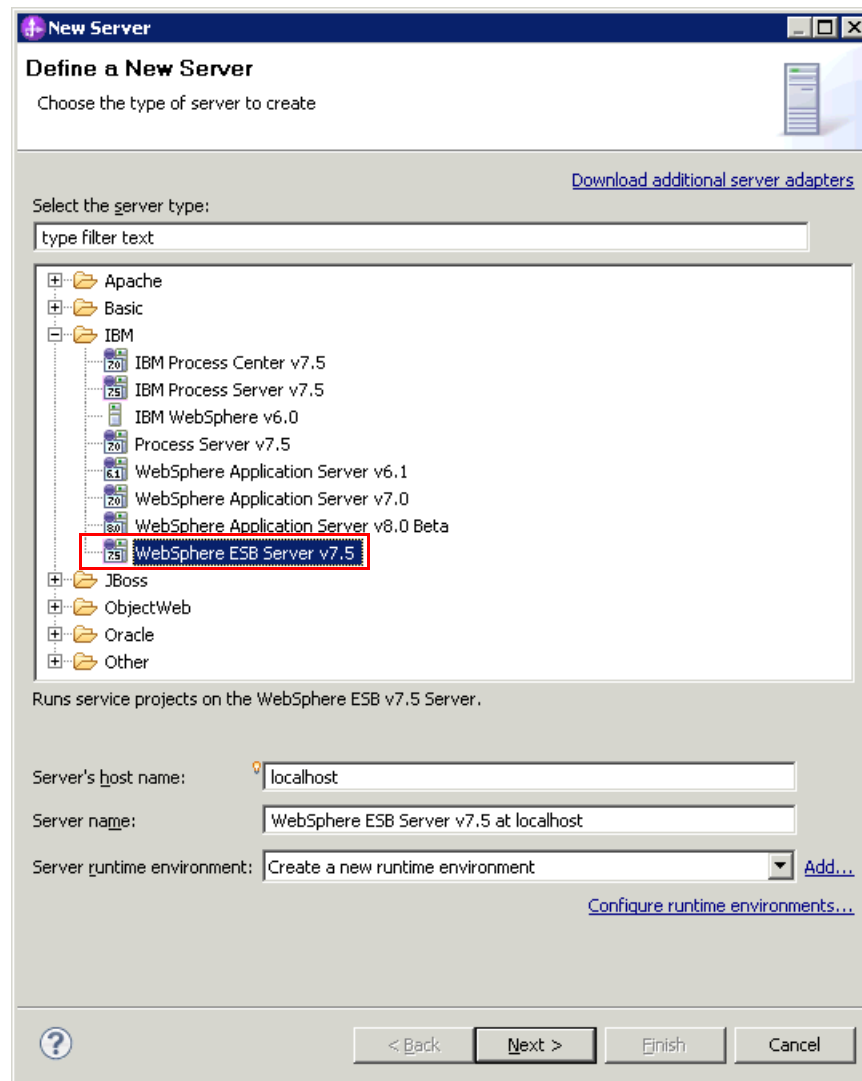


Figure 4-33 Define a new server

5. Specify the WebSphere Application Server Runtime Environment with a unique name and the installation directory of the WebSphere Application Server, as shown in Figure 4-34. Click **Next**.

Installation note: The installation process shown in 4.4, “Installing and configuring the runtime components” on page 81 installs the runtime environment separately from Integration Designer. When you install Integration Designer, runtime stubs with default names for the test environments are created. Therefore, to avoid collisions in our scenario, we set **WebSphere ESB Server 7.5 (1)** as the name of the new test environment.

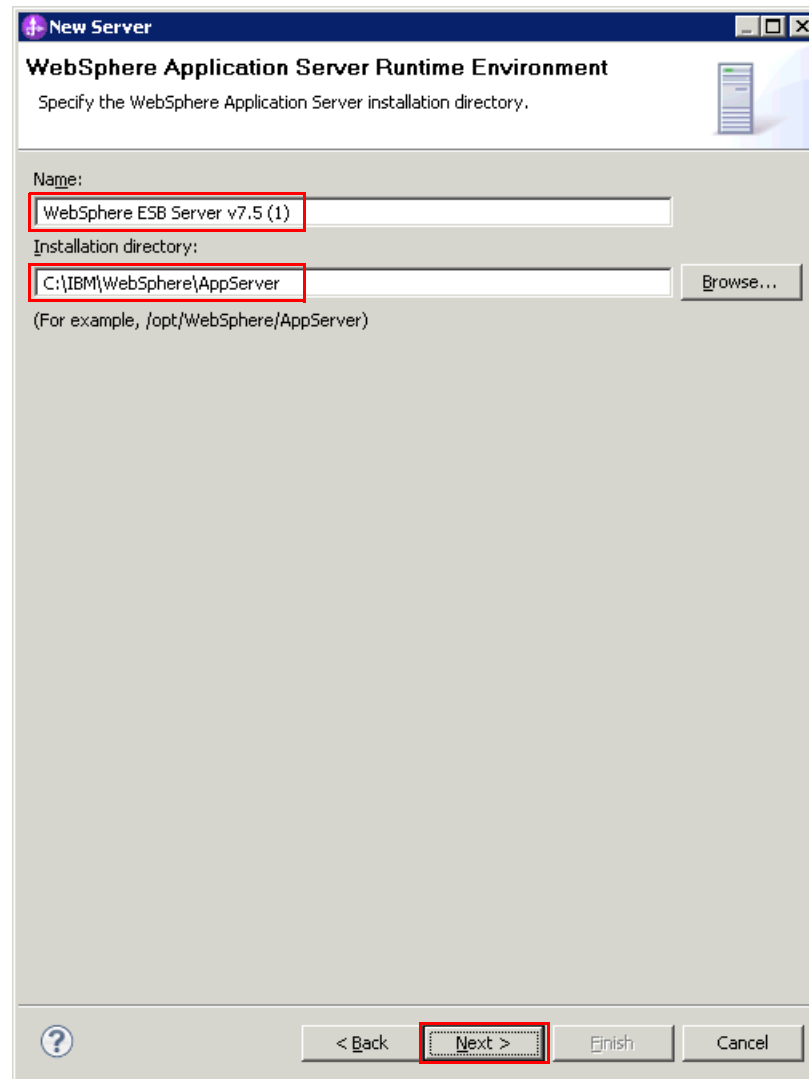
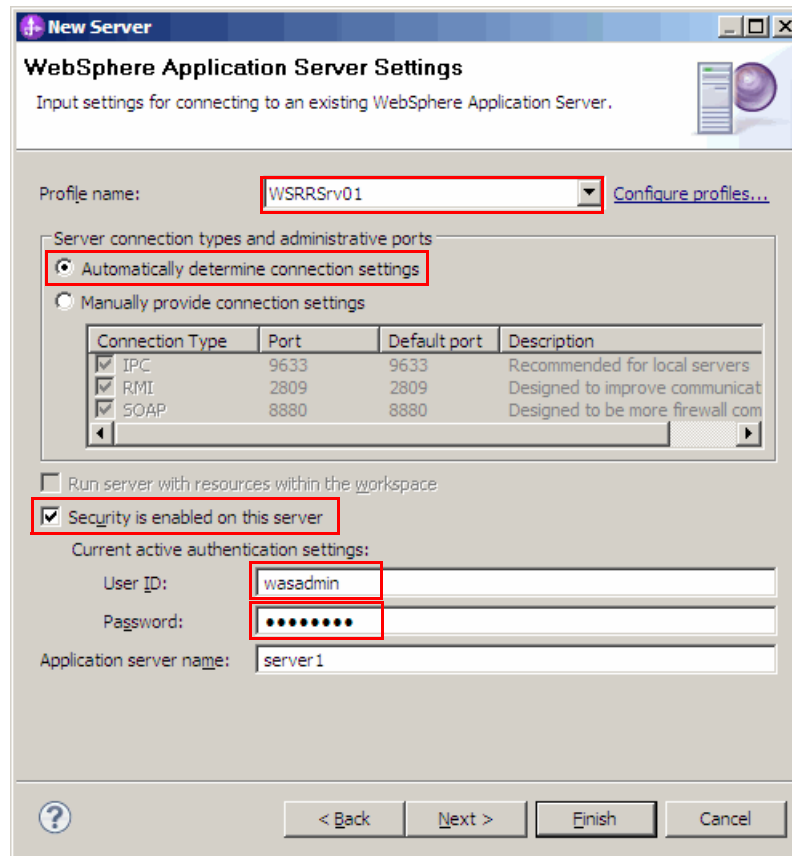


Figure 4-34 WebSphere Application Server Runtime Environment

- Next, provide the WebSphere Application Server input settings, such as a profile name and authentication data for enabling security on the server, as shown in Figure 4-35. Leave the default values for other settings, and click **Finish** to create the test environment definition.



The image shows the 'New Server' dialog box for 'WebSphere Application Server Settings'. The title bar says 'New Server'. The main title is 'WebSphere Application Server Settings' with a subtitle 'Input settings for connecting to an existing WebSphere Application Server.'.

Profile name: [Configure profiles...](#)

Server connection types and administrative ports

☒ Automatically determine connection settings

☐ Manually provide connection settings

Connection Type	Port	Default port	Description
<input checked="" type="checkbox"/> IPC	9633	9633	Recommended for local servers
<input checked="" type="checkbox"/> RMI	2809	2809	Designed to improve communication
<input checked="" type="checkbox"/> SOAP	8880	8880	Designed to be more firewall compliant

☐ Run server with resources within the workspace

☒ Security is enabled on this server

Current active authentication settings:

User ID:

Password:

Application server name:

Buttons: ? < Back Next > Finish Cancel

Figure 4-35 WebSphere Application Server Settings

- After completing this step the new test environment is created and the status of this server is *Stopped*, as shown in the Servers tab in Figure 4-36.

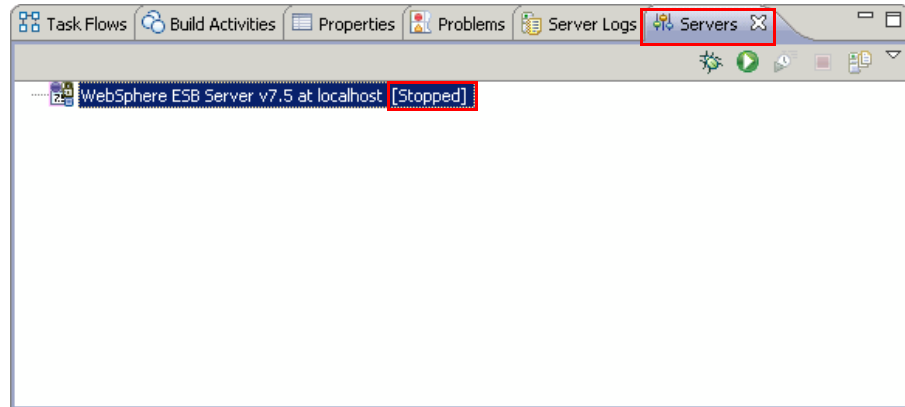


Figure 4-36 New test environment WebSphere ESB Server 7.5 at localhost (Stopped)

- Before starting the server, go to the “Server configuration setting” panel by double-clicking the server item in the Servers tab. The server configuration for “WebSphere ESB Server 7.5 at localhost” opens. Clear the **Start server with a generated script** option, as shown Figure 4-37.

Save the changes, and go back to the Servers tab to start the server. Click **Start the Server** on the upper-right corner of the Servers tab.

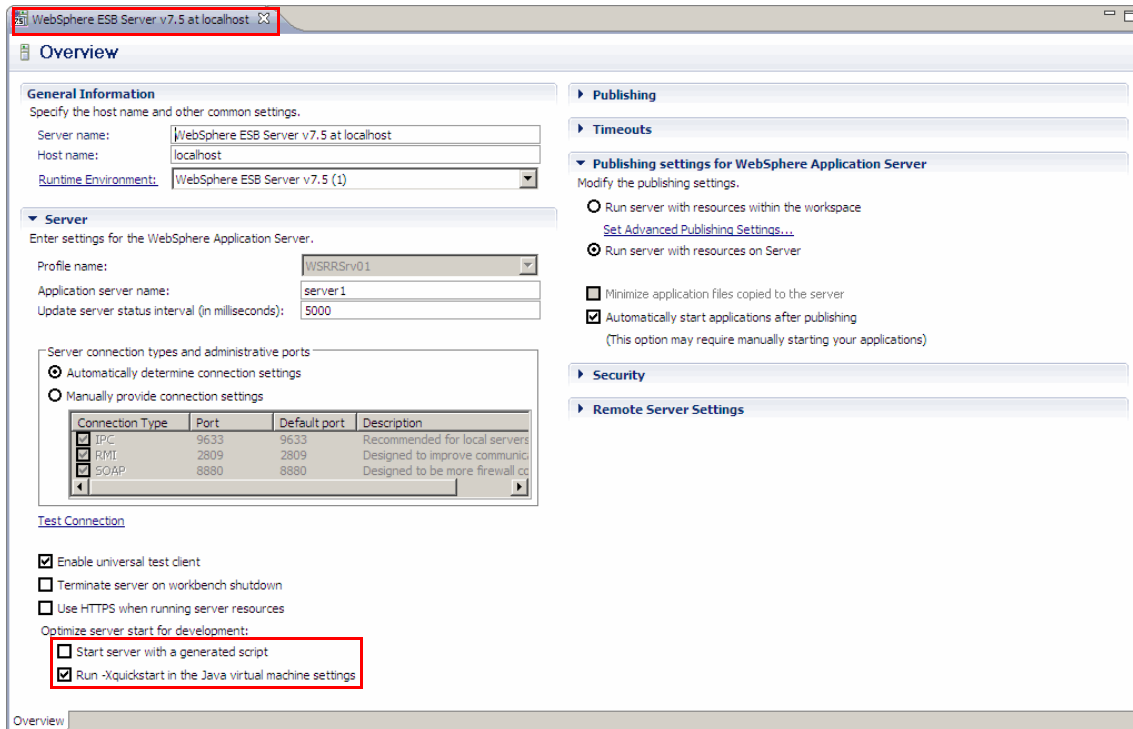


Figure 4-37 Configuration for WebSphere ESB Server 7.5 at localhost

9. After a successful start of the server the status is *Started, Synchronized* as shown in Figure 4-38.

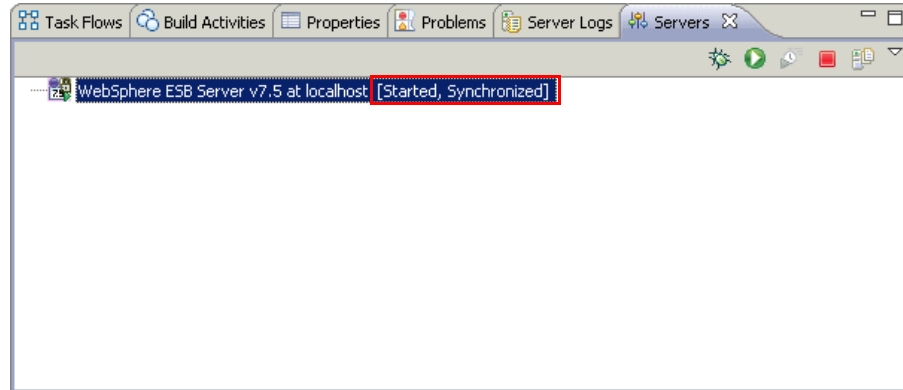


Figure 4-38 New test environment WebSphere ESB Server 7.5 at localhost (Started)

This test environment is the basis for test activities in this book.

4.5.2 Installing and configuring the WSRR Eclipse Plug-in

The WSRR Eclipse Plug-in provides the finding and publishing capabilities for Eclipse-based development environments. In this section, we discuss the following tasks:

1. Installing the WSRR Eclipse Plug-in
2. Defining a target server for the WSRR Eclipse Plug-in

You can find additional information about how to install and configure the WSRR Eclipse Plug-in at:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/twsr_eclipse_plugin_install.html

Installing the WSRR Eclipse Plug-in

This section explains how to install the WSRR Eclipse Plug-in on an existing Integration Designer installation. The WSRR Eclipse Plug-in is delivered as a compressed file, and you can find it on an existing WSRR installation. Install the WSRR Eclipse Plug-in using the standard installation process provided by the Eclipse environment:

1. Open Integration Designer, and go to **Help** → **Install New Software**. In the Eclipse software installer dialog box, click **Add** (Figure 4-39).

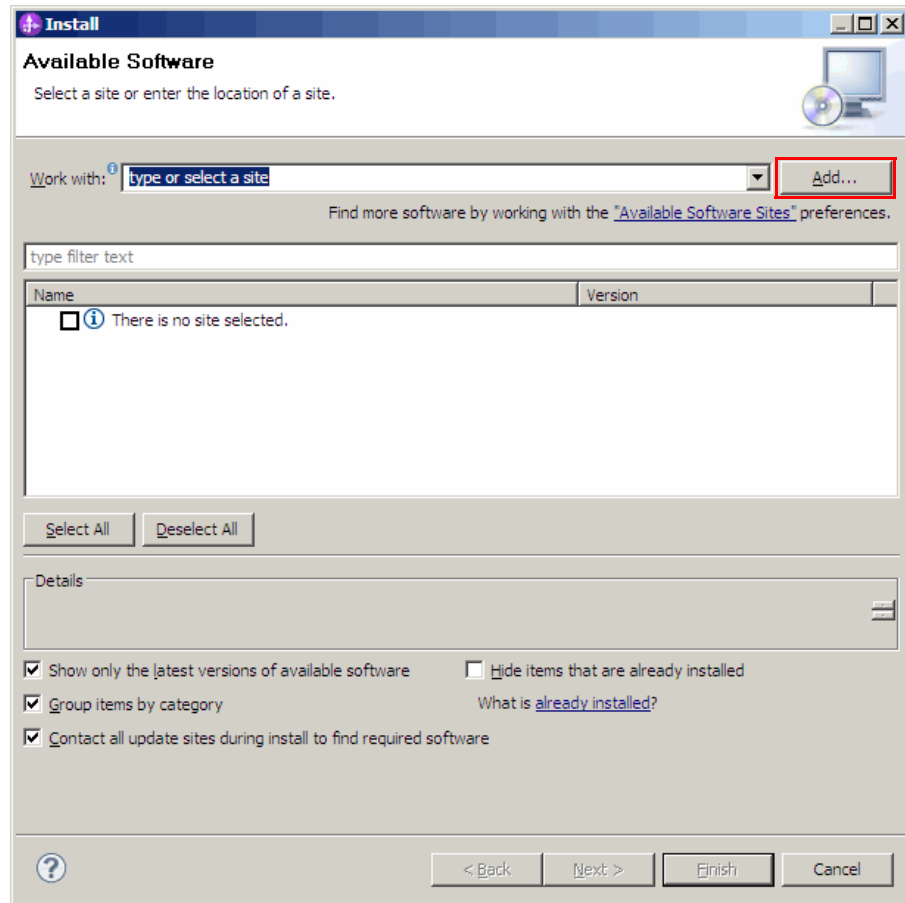


Figure 4-39 Eclipse software installer

2. In the Add Repository dialog box, click **Archive**, and select the WSRR_Eclipse_OSGi_ArchiveSite.zip file that is located in the %APPSERVER_ROOT%\WSRR folder (Figure 4-40). Click **OK**.

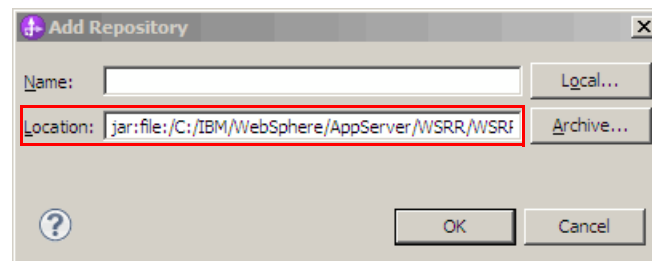


Figure 4-40 Add repository

3. The WSRR Eclipse Plug-in item is now available in the list of available software, as shown in Figure 4-41. Make sure that the item is selected, and click **Next**.

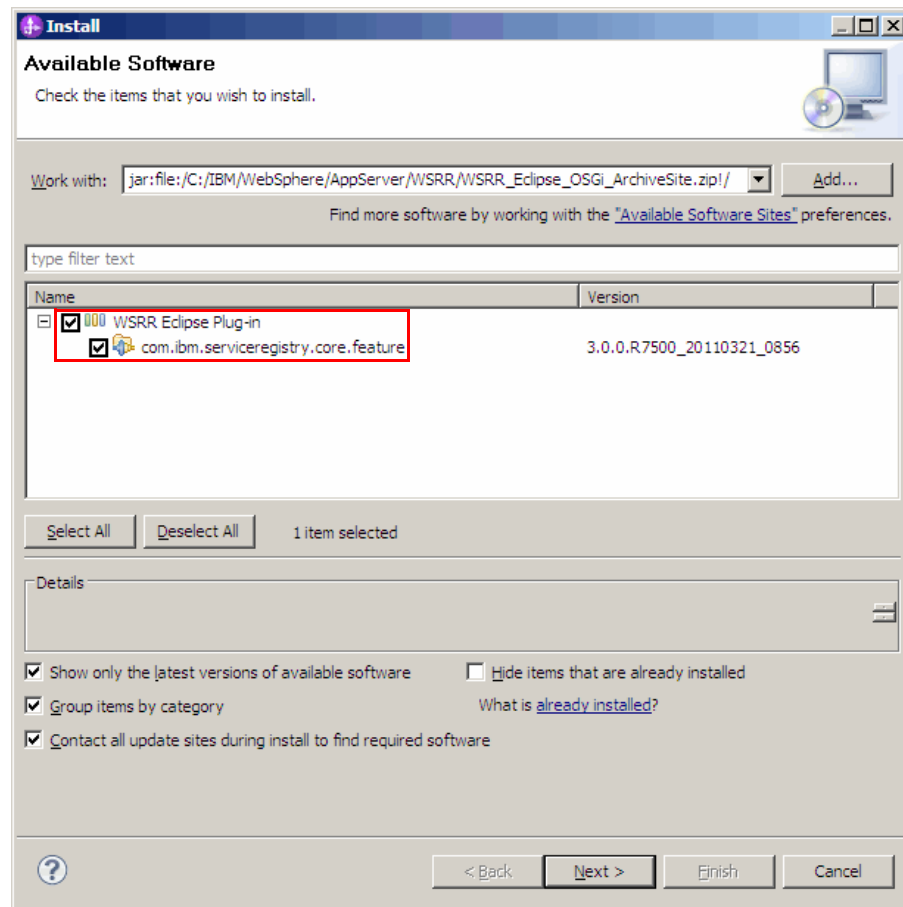


Figure 4-41 Eclipse software installer with WSRR Eclipse Plug-in

4. In the Install Details panel, review the items to be installed, and click **Next**.
5. Finally, review and accept the license agreement, and click **Install**. The installation task begins, showing a progress bar.

Installation note: A Security Warning might display during the installation task, alerting you about unsigned content. If this warning displays, click **OK** to continue.

6. After a few minutes, a notification informs you that the installation is completed (Figure 4-42). Click **Restart Now** to apply the changes and to proceed to the next task.

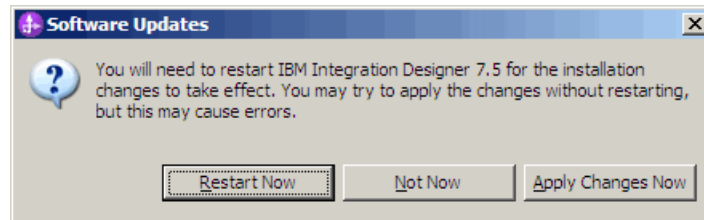


Figure 4-42 Installation completed

Defining a target server for the WSRR Eclipse Plug-in

In this section, we explain how to create a WSRR location for the WSRR Eclipse Plug-in. A WSRR location holds information about a WSRR instance on which you want to search or publish services information.

Workspace note: WSRR locations are stored at the workspace configuration level. Therefore, you must complete the following steps again when switching to another workspace location.

To configure a WSRR location:

1. Open Integration Designer. Go to **Window** → **Open Perspective** → **Other**, and choose **Java** from the possible list entries as shown in Figure 4-43. Click **OK**.

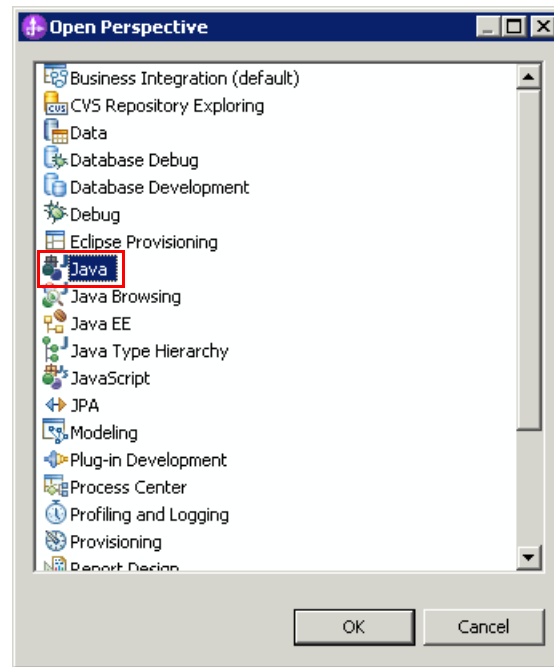
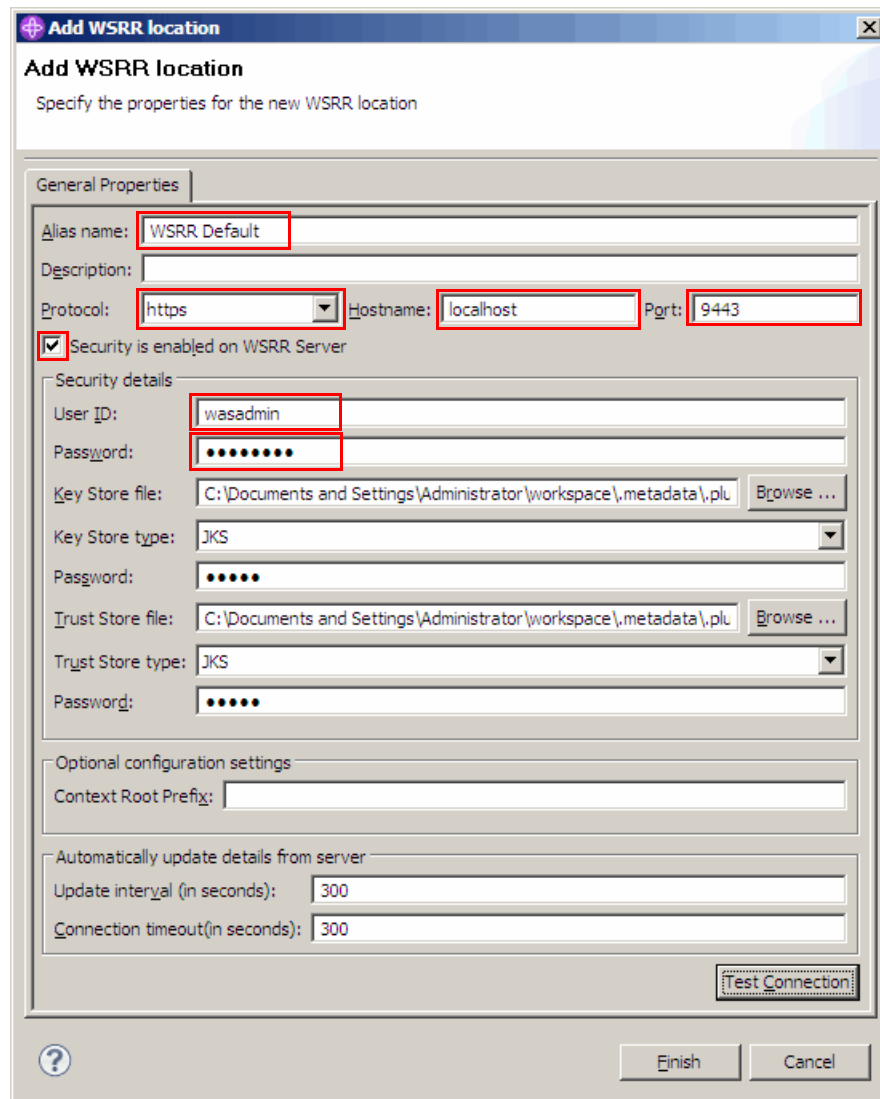


Figure 4-43 Open Perspective Java

-
- The screenshot shows the 'WebSphere Service Registry and Repository (WSRR)' Preferences dialog box. On the left, a tree view lists various configuration categories, with 'WebSphere Service Registry and Repository (WSRR)' selected and highlighted by a red rectangle. The main area on the right is titled 'WSRR Preferences' and contains a text box with instructions: 'Add, remove or edit WSRR location information. The checked location will, by default, be used for all communications with a WSRR instance, unless otherwise prompted.' To the right of this text are three buttons: 'Add ...' (highlighted with a red rectangle), 'Edit ...', and 'Remove'. Below the text is a table with columns 'Name', 'WSRR Host', and 'Description'. The table is currently empty. At the bottom of the dialog are 'Restore Defaults' and 'Apply' buttons. The very bottom of the window shows 'OK' and 'Cancel' buttons.

122 Smart SOA Solutions with WebSphere Enterprise Service Bus Registry Edition V7.5

4. Provide an alias name and modify the connection and security options according to the server instance previously deployed, as indicated in Figure 4-45. Click **Test Connection** to verify that the connection configuration is valid.



The image shows a Windows-style dialog box titled "Add WSRR location". Below the title bar, the text "Specify the properties for the new WSRR location" is displayed. The dialog has a tabbed interface with "General Properties" selected. The fields are as follows:

- Alias name: "WSRR Default" (highlighted with a red box)
- Description: (empty)
- Protocol: "https" (dropdown menu, highlighted with a red box)
- Hostname: "localhost" (text box, highlighted with a red box)
- Port: "9443" (text box, highlighted with a red box)
- ☒ Security is enabled on WSRR Server
- Security details section:
 - User ID: "wasadmin" (text box, highlighted with a red box)
 - Password: (masked with dots, highlighted with a red box)
 - Key Store file: "C:\Documents and Settings\Administrator\workspace\metadata\plu" (text box) with a "Browse ..." button
 - Key Store type: "JKS" (dropdown menu)
 - Password: (masked with dots)
 - Trust Store file: "C:\Documents and Settings\Administrator\workspace\metadata\plu" (text box) with a "Browse ..." button
 - Trust Store type: "JKS" (dropdown menu)
 - Password: (masked with dots)
- Optional configuration settings section:
 - Context Root Prefix: (empty text box)
- Automatically update details from server section:
 - Update interval (in seconds): "300" (text box)
 - Connection timeout(in seconds): "300" (text box)

At the bottom right, there is a "Test Connection" button. At the bottom left, there is a help icon (?). At the bottom right, there are "Finish" and "Cancel" buttons.

Figure 4-45 Add WSRR location dialog box

Attention: Make sure that the server and the service registry application are running *before* you test the connection.

5. When you attempt to connect to the WSRR server for the first time, you receive the warning **Signer certificate not found**. Click **Accept** to import the necessary SSL signer certificate into the trust store for this WSRR location.
6. When notified about the results of the test, as shown in Figure 4-46, click **OK**.

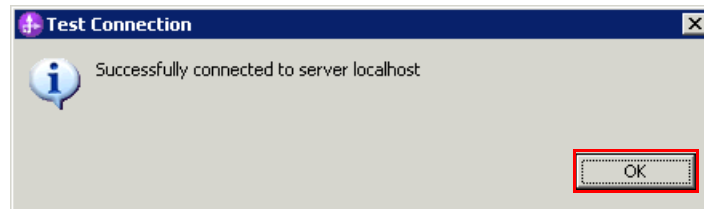


Figure 4-46 Test connection results

7. Click **Finish** to add the WSRR location to the WSRR Eclipse Plug-in. Close the Preferences window by clicking **OK**.

4.5.3 Installing and configuring WSRR Studio

Note: The installation and configuration of WSRR Studio is not required for the scope and scenarios covered in this book. However, for completeness, we describe these tasks in this section.

WSRR Studio is a customizing tool for WSRR. In this section, we describe the following tasks:

1. Installing WSRR Studio.
2. Configuring a connection to an existing server.

You can find additional information about how to install and configure WSRR Studio at:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.studio.doc/wsrr_studio_homepage.html

Installing WSRR Studio

WSRR Studio includes its own version of the Launchpad tool, which we used in previous sections. To install this component, use the Launchpad tool as follows:

1. Locate and run the `Launchpad.exe` file (or the corresponding file if your platform is not based on the Windows operating system) for WSRR Studio.

The Launchpad tool allows you to perform several tasks, as shown in Figure 4-47. Click **Install WebSphere Service Registry and Repository Studio** to continue.

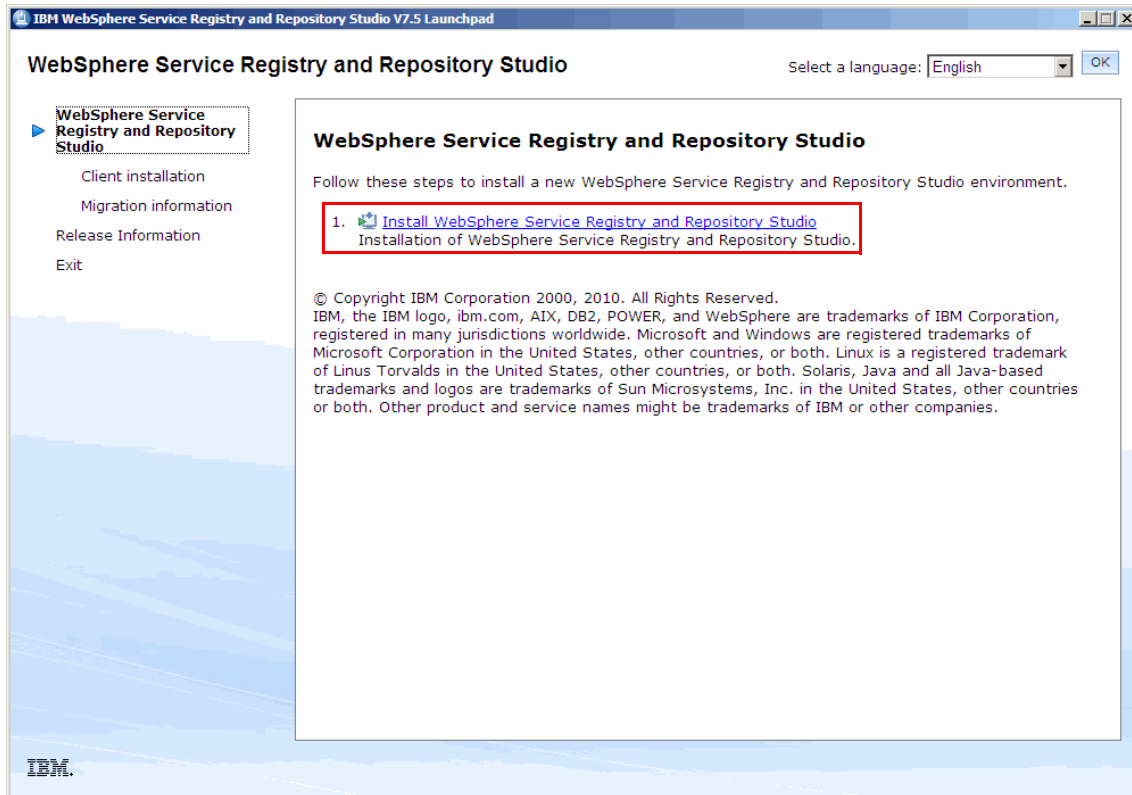


Figure 4-47 WSRR Studio Launchpad welcome panel

2. Select the packages to install as indicated in Figure 4-48, and click **Next**.

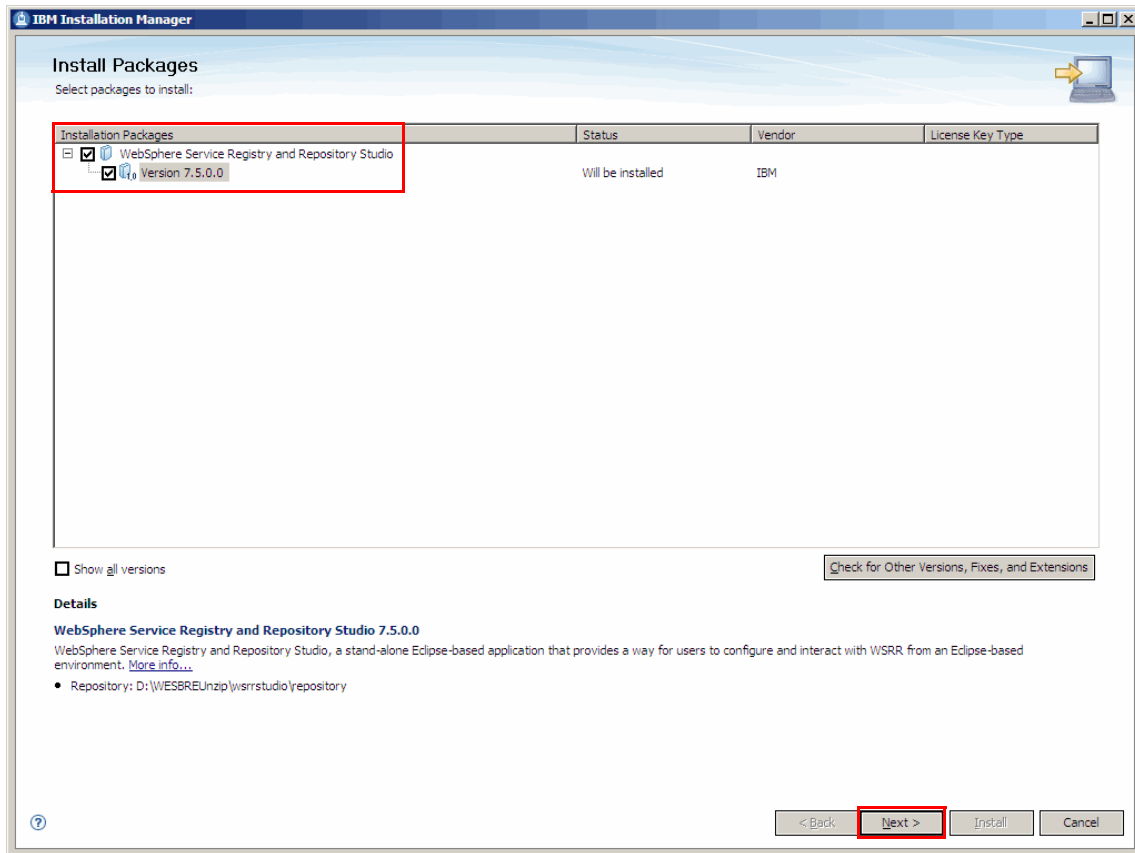


Figure 4-48 Package selection

3. Read and accept the license agreement, and click **Next**.
4. Select **Create a new package group**. Review the package group installation directory, and modify it if needed. Click **Next**.
5. The “Select features” panel allows you to decide which subcomponents to include in this installation. In our scenario, there is only one main component and no available subcomponents. Click **Next**.
6. Finally, review the summary information, and click **Install** to initiate the installation task.

7. The process takes from 10 to 30 minutes, depending on the performance of the target system. When the installation is complete, a message displays as shown in Figure 4-49. Click **Finish** to close the installation window.

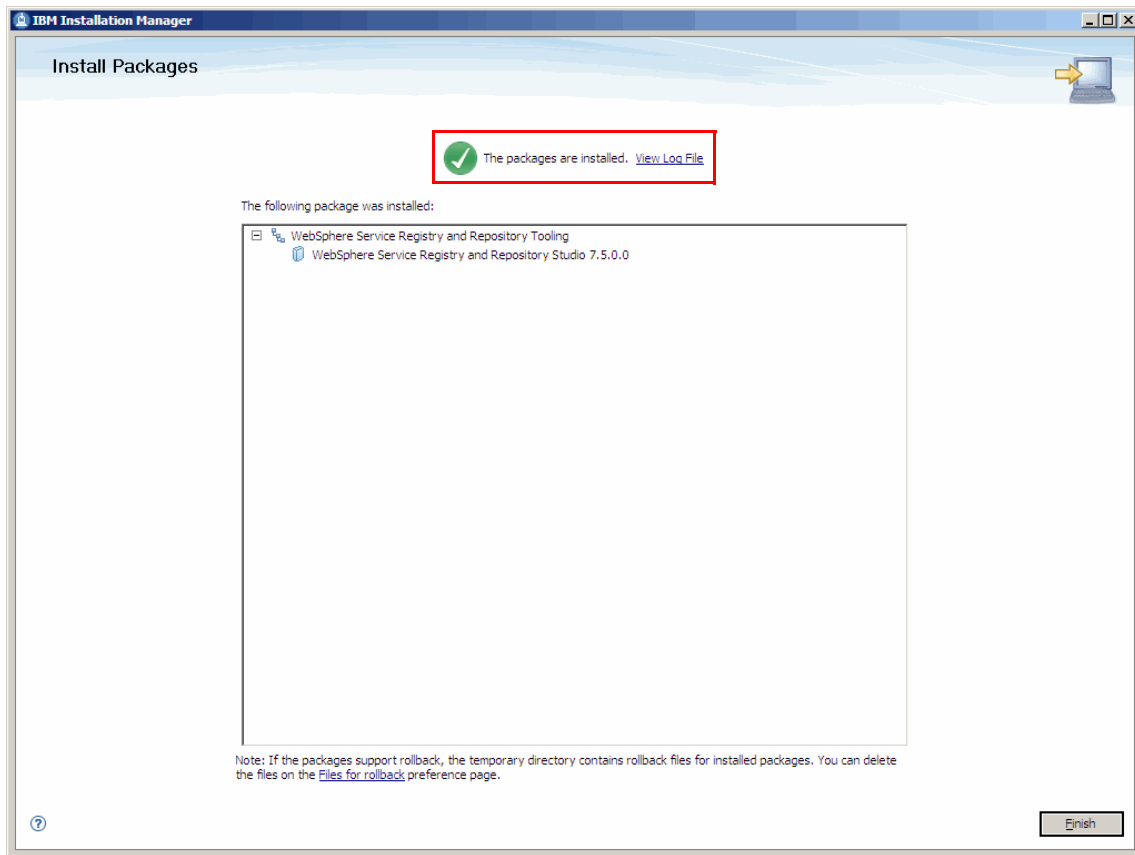


Figure 4-49 Installation results panel

Configuring a connection to an existing server

This section explains how to create a WSRR location for WSRR Studio. A WSRR location holds information about a WSRR instance on which you want to work with. To configure the WSRR location:

1. Open WSRR Studio, and select the WSRR Locations tab. Right-click the blank canvas, and click **Add WSRR Location**, as shown in Figure 4-50.

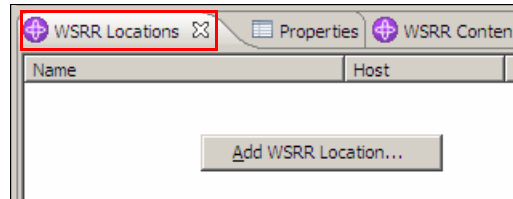
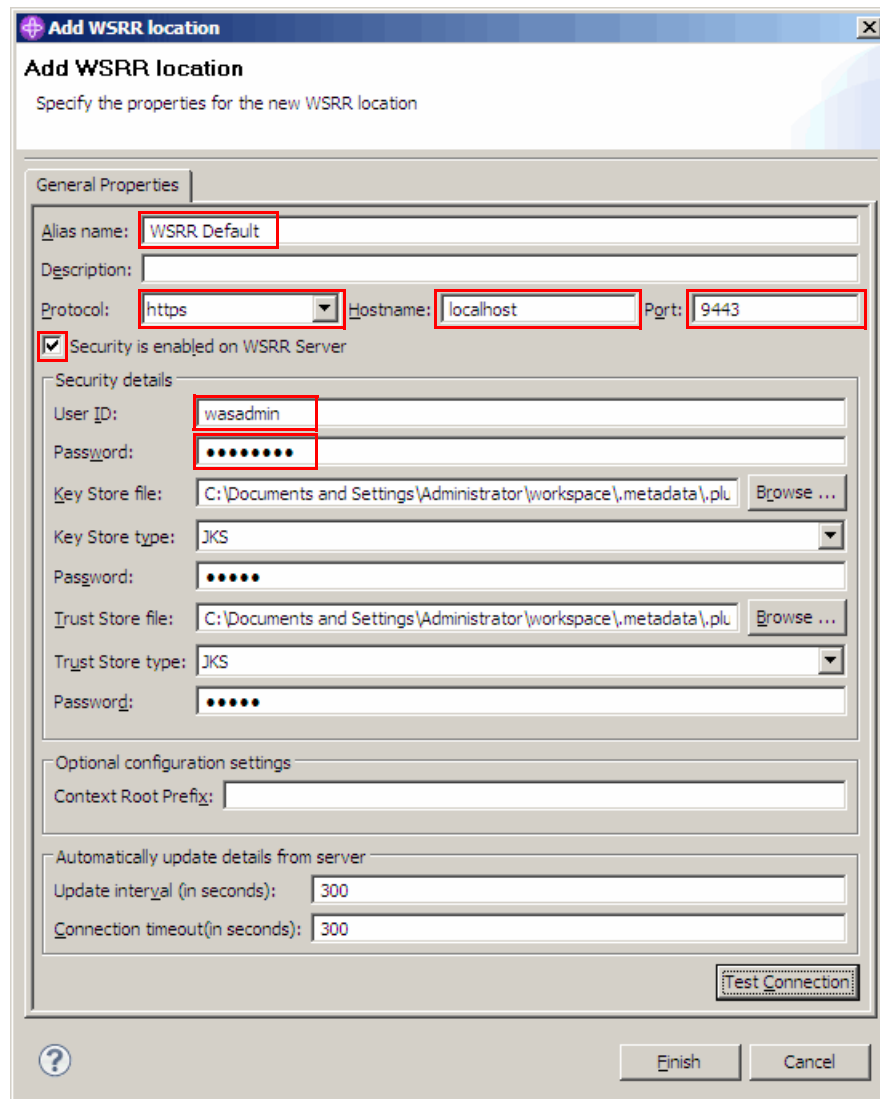


Figure 4-50 WSRR Locations tab

2. Provide an alias name and modify the connection and security options according to the server instance previously deployed, as indicated in Figure 4-51. Click **Test Connection** to verify that the connection configuration is valid.



The image shows a Windows-style dialog box titled "Add WSRR location". Below the title bar, the text "Specify the properties for the new WSRR location" is displayed. The dialog is divided into several sections. The "General Properties" section contains fields for "Alias name" (set to "WSRR Default"), "Description", "Protocol" (set to "https"), "Hostname" (set to "localhost"), and "Port" (set to "9443"). A checkbox labeled "Security is enabled on WSRR Server" is checked. Below this is the "Security details" section, which includes fields for "User ID" (set to "wasadmin"), "Password" (masked with dots), "Key Store file" (set to "C:\Documents and Settings\Administrator\workspace\metadata\plu"), "Key Store type" (set to "JKS"), "Trust Store file" (set to "C:\Documents and Settings\Administrator\workspace\metadata\plu"), "Trust Store type" (set to "JKS"), and another "Password" field (masked with dots). The "Optional configuration settings" section includes a "Context Root Prefix" field. At the bottom of the dialog, there is a "Test Connection" button, a help icon (?), and "Finish" and "Cancel" buttons.

Figure 4-51 Add WSRR location dialog box

Attention: Make sure that the server and the service registry application are running *before* you test the connection.

- When you attempt to connect to the WSRR server for the first time, you receive the warning shown in Figure 4-52. Click **Accept** to import the necessary SSL signer certificate into the trust store for this WSRR location.

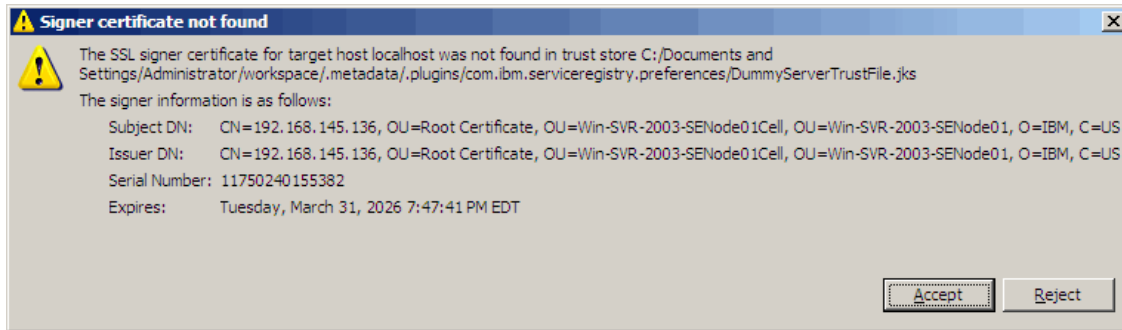


Figure 4-52 Importing the SSL signer SSL certificate

- When notified about the results of the test, as shown in Figure 4-53, click **OK**.

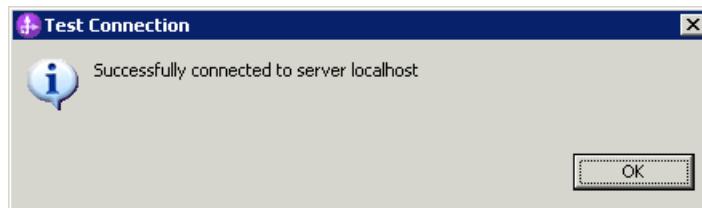


Figure 4-53 Test connection results

- Click **Finish**. The WSRR location now displays in the WSRR Locations tab, as shown in Figure 4-54.

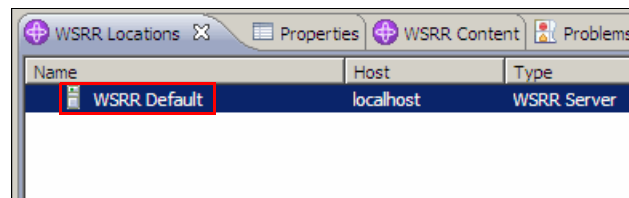



Figure 4-54 WSRR Locations tab with a new entry



Part 3

Building WebSphere Enterprise Service Bus Registry Edition solutions



The case study and scenario used in this book

This book uses a fictitious supply company to illustrate how to adopt SOA as the architecture to implement and govern the entire integration architecture using WebSphere ESB Registry Edition. For this company, WebSphere ESB Registry Edition serves as the flexible enterprise service bus that is governed by an authoritative registry and repository.

This chapter introduces the case study and provides the background of the organization's needs and goals. It illustrates the chosen service-oriented architecture (SOA) governance solution and its value to the organization. It also describes how several requirements change at various points in the history of the company.

5.1 Introduction to the case study

The supply company in this case study faces the challenge of continuous change in the business environment, which mandates a high degree of flexibility in the IT environment. The supply company has to continuously adapt the existing operational IT services for the business changes. Being a large and complex organization, it also needs to govern and enforce policies for the services and their changes.

The supply company has the following business and IT management goals:

- ▶ Enable business agility by creating a dynamic IT infrastructure
- ▶ Create governed and assured IT services
- ▶ Reduce the cost and complexities of integration

5.2 The supply company's SOA vision and requirements

The supply company's strategic vision is to continuously evolve into new areas of sustainable and quality business growth. It aspires to be counted as one of the most adaptive business organizations, a company that reinvents itself into a leadership position across business cycles. The supply company's IT management envisions that the IT organization can encompass flexibility and dynamicity as core principles, thus becoming a key enabler for business agility.

The supply company's current IT landscape consists of a variety of custom and packaged solutions serving various business units and functions. These solutions and applications are integrated through a number of mechanisms, including integration and service platforms.

The supply company understands that SOA is an effective technique for developing software applications that best align with business models. However, SOA requires an increased level of cooperation and coordination between business and IT, and among IT departments and teams. It can be challenging to ensure that all parties are working towards the common goal to align business and IT, plan better reuse, and create a higher return on investment (ROI).

To ensure this coordination between teams, it is important to define which decisions are taken by whom and how these decisions are made during the services development life cycle. SOA governance ensures that this decision-making model is defined. SOA governance is a specialization of IT governance that puts the key IT governance decisions within the context of the life cycle of service components, services, and business processes.

Figure 5-1 shows the SOA governance model.

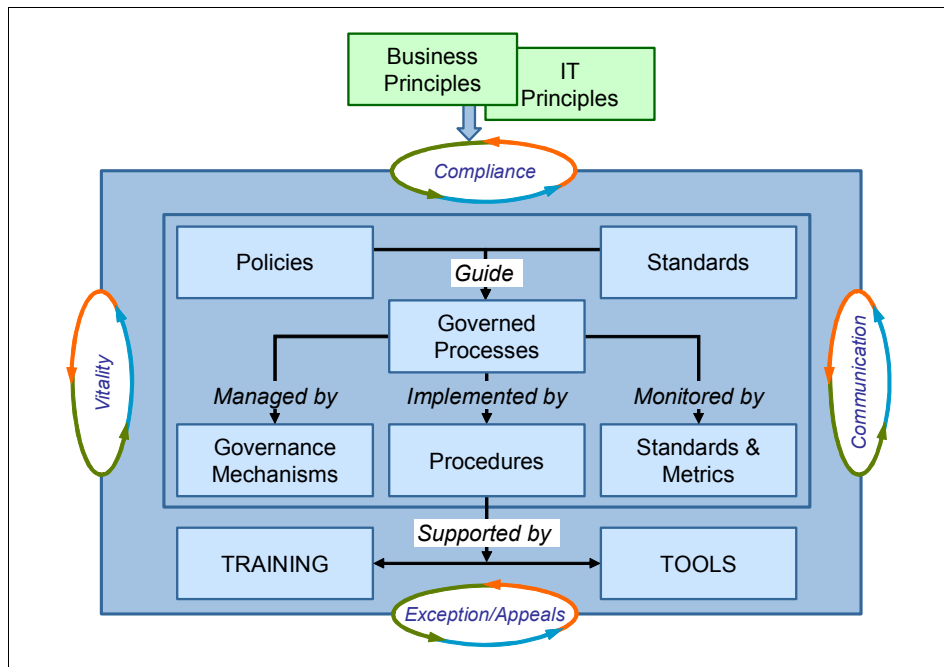


Figure 5-1 Mechanics of a governance model

To plan, guide, and implement an effective and well-governed SOA throughout the enterprise, the supply company set up an SOA Center of Excellence (CoE). This CoE contains a group of expert practitioners who establish policies for identification and development of services, establishment of SLAs, management of registries, and other efforts that provide effective governance. The CoE members also put those policies into practice by mentoring and assisting teams with developing services and composite applications.

Figure 5-2 shows the structure of the CoE.

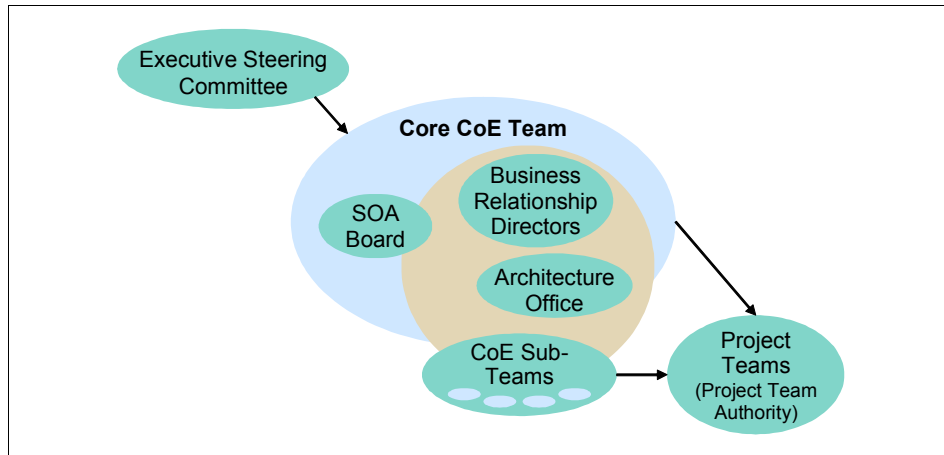


Figure 5-2 The supply company SOA CoE structure

The SOA CoE decided that service identification, design, and development of services will follow a life cycle, which IBM calls the *SOA life cycle*. Similarly, SOA governance also follows the *SOA governance life cycle*. These two life cycles working together ensure that an effective and governed SOA is implemented.

Figure 5-3 shows the relationship between the two life cycles.

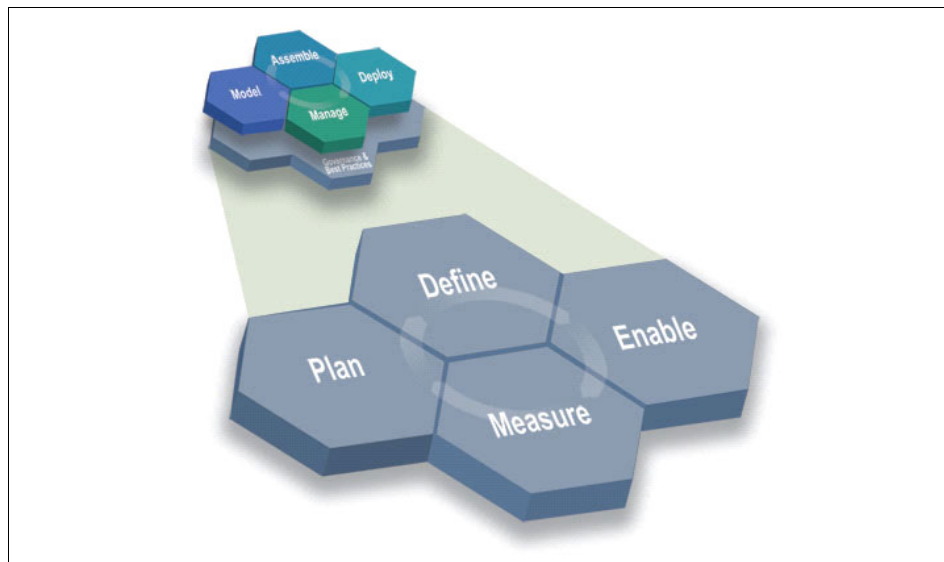


Figure 5-3 SOA life cycle and SOA governance life cycle

Based on the mandate from the executive steering group to adopt a robust and well-governed SOA, the following primary goals are defined for the SOA CoE architecture review board:

- ▶ Create a standards-based enterprise service bus
- ▶ Enable flexible and dynamic service mediation
- ▶ Ensure governance and policy enforcement
- ▶ Promote service reuse with control

The platform allows various functional requirements to be fulfilled as part of the SOA life cycle. For example, the required services are discovered and identified using various techniques, such as analysis of business domains, existing assets, and business goals of the supply company. Also, as part of the SOA life cycle, after the candidate services are identified, they go through a rigorous process of filtering, based on business and architecture priorities, reuse potential, ROI, and so on, to arrive at the services portfolio. The services are then implemented and deployed to the platform.

The fulfilment of non-functional requirements for these services and the platform as a whole are ensured by a well-defined and well-executed SOA governance life cycle. To meet the SOA CoE primary goals and to define an appropriate governance model for the supply company, detailed non-functional requirements (NFRs) are specified for each of the goals. We describe these NFRs in the sections that follow.

5.2.1 Create a standards-based enterprise service bus

The first step in creating a well-governed SOA from an architectural point of view is to adopt an ESB and registry combination. All services must be gradually visible and deployed on this platform, and other point-to-point or custom integration points must be removed. This combination of an ESB and registry leads to the creation of a platform on which the enterprise can rely as the system of record for services, allowing for a systematic approach to management and governance.

The following NFRs are defined to achieve this goal:

- ▶ NFR-101 - The supply company wants to build an ESB to provide a standardized connectivity layer that enables flexible and simple integration of services to clients, thereby decoupling integration logic from applications and avoiding point-to-point integration.
- ▶ NFR-102 - The supply company wants the ESB platform to allow for easy governance of services, such that all stakeholders can rely on the platform to be the single system of record for enterprise services.

5.2.2 Enable flexible and dynamic service mediation

The supply company understands from its own and industry experience that even when an integration platform is used, one of the frequent change cases is the need to change the link between the integration platform and service providers. The supply company understands that it must create the integration platform in such a way that this coupling between the integration platform and service providers is easy to change.

The following NFRs are defined to achieve this goal:

- ▶ NFR-201 - The coupling between integration logic on the integration platform and service providers endpoints must not be embedded within the integration logic. This coupling should be externalized in the form of a registry lookup, making the coupling easier to change with changing business requirements. The supply company wants flexibility when changing service implementation or switching to newer versions.
- ▶ NFR-202 - The supply company wants service clients to be decoupled from the service they consume. Thus, if the service is moved to another location, the supply company does not need to inform the service clients. The supply company wants these changes to be transparent to the service clients.
- ▶ NFR-203 - To have visibility into who is actually using their service, the supply company wants to introduce service level agreements (SLAs) between the service clients and the services providers. Based on these agreements, the supply company can check at run time whether a client is allowed to use the requested service.

5.2.3 Ensure governance and policy enforcement

The supply company runs through multiple cycles and iterations of the SOA life cycle and SOA governance life cycle. After completing such a cycle, the enterprise is considered mature enough to introduce policy-driven integrations and enforcements.

The following NFRs are defined to achieve this goal:

- ▶ NFR-301 - To improve flexibility and maintainability in the integrations flows, all mediations flows must be as flexible as possible, as long they do not affect other non-functional requirements. The integrations must use defined policies such that they are aligned to the business change cases and that they ensure uniform compliance.
- ▶ NFR-302 - The ESB and registry forms the foundation of the critical services platform and must be scalable to be used by a large number of consumers. It is critical that interactions between the ESB and registry are optimized and

that overheads are eliminated, where possible. To avoid multiple interactions between the ESB and the registry, when there are two or more ESB components that interact with WSRR, group these components to get only one interaction while also balancing the overhead on ESB and the performance in WSRR responses.

- ▶ NFR-303 - To improve flexibility, if business rules can be represented as policies in the WSRR, then mediations flows must be designed in such a way that changes to these business rules do not impact the implementations of the mediation flow or the consumer or the service but only require configuration changes in policies. For example, in one of the typical services at the supply company, the Account Creation service is required to create a policy to dynamically change the flow configuration based on the customer's country.

5.2.4 Promote service reuse with control

The supply company understands that a frequent constraint to service reuse is insufficient support for controlled versioning. This constraint leads to a high risk assessment for existing users with respect to any initiative to change a service for requirements of new users. This frequently leads to the creation of similar services that should be a single service. Without an ability to implement the changes discovered later, it is normally not possible to promote reuse of a service.

The following NFR is defined to achieve this goal:

- ▶ NFR-401 - The platform must support governed evolution of a service. It is quite common that all requirements for a particular service do not get captured during the initial design and rollout of a service. The platform must allow the service to evolve and change with enough control and minimal impact to existing users.

5.3 The supply company's SOA governance strategy

The SOA architecture review board determined that the supply company needed a roadmap for its SOA governance journey to enable flexibility with control. Learning from experiences in the industry and its own experiences with integration and SOA-based initiatives in fragmented departments and units, the supply company realized that including the concept of *governed* services at the beginning of the process is critical.

The SOA architecture review board also determined that to secure ROI, to decrease risks, and to continuously improve upon and align with business

requirements and priorities, it was important to build and roll out the initiative incrementally.

The supply company understood that the SOA life cycle and SOA governance life cycle would be executed cyclically across incremental phases, such that the benefits of SOA and a governed platform are continuously realized and ROI is secured. For each of the incremental phases, the SOA life cycle and SOA governance life cycle will be executed with expanding and evolving scope, priorities, and business needs.

Figure 5-4 shows the phases of the SOA journey.

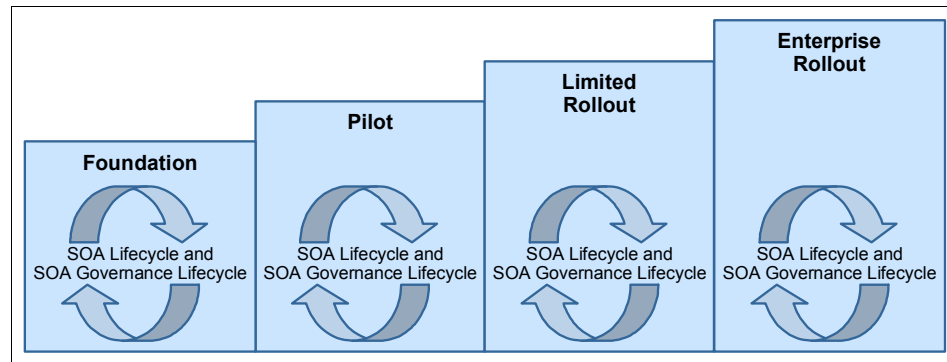


Figure 5-4 The supply company's SOA journey

After defining this rollout strategy, the supply company decided to use a set of a few services as the initial exemplar to demonstrate flexibility to changes, benefits of dynamic mediations, and enforcement of governance mechanisms such as policies and SLAs. This demonstration of value of a governed services platform is crucial for the supply company to roll out services on an enterprise scale.

The supply company decided to use WebSphere ESB Registry Edition as the standards-based enterprise bus. WebSphere ESB Registry Edition has strong support for embedded governance and service dynamicity.

After discussions with the key stakeholders, the Account Creation service was chosen as the exemplar. The service will be a reusable service that can flow through the service life cycle governance process as defined in WebSphere ESB Registry Edition, from the proposed state to the final realization and deployment to the enterprise bus.

5.3.1 Foundation phase

The foundation phase introduces the concept of governed services and the establishment of a platform that enables runtime flexibility and enforces runtime compliance to policies. During this phase, the SOA CoE was formed and the initial SOA governance goals and practices were formally specified and defined into the WebSphere ESB Registry Edition platform.

5.3.2 Pilot phase

The Account Creation service is the exemplar for the initial cycle of the SOA Governance journey. One line of business (LOB) of the supply company agreed to be the sponsor for enabling this service into the governed services platform. The service provider Enterprise Resource Planning (ERP) application agreed to participate in this initiative.

The Account Creation service was proposed and passed through the approval processes. Then, the implementation and deployment was completed by the services design and development team. The consuming LOB began to use this service using the WebSphere ESB Registry Edition-based governed enterprise services platform. The objective of this phase was to prove that the governance process that was defined by the supply company can be enforced and is usable by various actors in the service governance life cycle.

5.3.3 Limited rollout phase

With the Account Creation service now available on the governed enterprise services platform, a few other LOBs wanted to use it as well. Also, changes were requested by the new business units to support more account types. These changes were now introduced to the service interface and a new version was created.

Without any direct impact to the existing consumer, the services platform switched the service request to the newer service version. During this phase, the usefulness of allowing changes to be deployed onto the service using a structured governed process and without causing any impact to existing consumers was deemed useful. Also, an ability to onboard the new consumers with SLA enforcement at run time was demonstrated. By this time, quite a few other services progressed through the SOA governance life cycle processes and were deployed. This set of services demonstrates the flexibility and value of the platform.

5.3.4 Enterprise-wide rollout phase

With the Account Creation service now being used by a few LOBs, and with other LOBs expressing their interest as well, there is a compelling business case to make the service completely enterprise-scale. By this time, a set of services with critical business units as users and providers was deployed onto the platform. Continuous assurance and compliance is now mandatory for the platform. Thus, mature governance practices and patterns would be rolled out, and SLAs and governance by policies enforced. The SOA governance principles were formally deployed using policies and enforced during run time by the platform.



Governance enablement profile

WebSphere Service Registry and Repository (WSRR) allows customization of entities (system and business models) and their properties, life cycles, the relationships among them, and classifications to represent a service environment. It provides role-based access control and user perspectives for stakeholders in various roles to perform governance tasks.

These artifacts constitute the governance model for your service environment. A configuration profile stores the entire set of configuration files (either .owl or .xml files) that describe various components of the governance model. A *profile* is packaged in a single compressed file, which facilitates backup, restore, and management of the entire governance model.

WSRR provides the following configuration profiles:

- ▶ Basic profile
- ▶ Governance enablement profile

Typically, you load and activate a configuration profile (a .zip file) of your choice into your WSRR configuration after completing the installation. This chapter provides a brief overview of the governance enablement profile. For more detailed information, refer to the product information center and *Service Lifecycle Governance with IBM WebSphere Service Registry and Repository*, SG24-7793.

6.1 Governance enablement profile overview

This section explains key concepts or entity models and relationships in the governance enablement profile. In particular, we concentrate on concepts related to service level agreements (SLAs) and service versioning.

The *governance enablement profile* is a WSRR profile that provides a starting set of configurations to manage services from the initial specification through the deployment in production in a service-oriented architecture (SOA) environment. It consists of the following components:

- ▶ **Models**
Entities that represent the objects in an organization's SOA environment
- ▶ **Roles and access control**
A predefined set of roles into which users can be assigned according to their SOA tasks
- ▶ **Life cycles**
A predefined set of states and transitions that are applied to artifacts to indicate their progress in the development process
- ▶ **Policies**
A predefined set of governance policies that control life cycle transition and decision rights, synchronization between life cycles, and validation of the requirements for metadata properties and relationships
- ▶ **Configuration of provided plug-in modifiers**
Configuration of modifier and notifier plug-ins provided by WSRR to enforce policies, validations, and correlations
- ▶ **Web user interface configurations**
A set of XML configuration files to configure and display registry content in various views and perspectives, based on WSRR roles, in its web user console, including business space configurations for different WSRR roles

In this chapter, we describe these components. We also discuss governance profile taxonomy and how to modify the governance enablement profile.

6.2 Models in the governance enablement profile

The governance enablement profile provides entities using *models* that can be used to represent the service description artifacts and metadata of an

organization's SOA environment. The governance enablement profile provides the following models:

- ▶ Service model

When Web Service Definition Language (WSDL), XML Schema Definition (XSD), and Service Component Architecture (SCA) modules and MQ artifacts are imported into WSRR, derived entities are created. These derived entities are represented as logical entities in the service model.

For each of the logical objects derived from the WSDL or XSD document, the correlator modifier creates a corresponding correlator object based on the definition of the conceptual service model entities. In addition, a custom relationship is created from the correlator service model entity to the logical entity.

- ▶ Governance enablement model

This model provides entities that can be used by an organization to represent business and SOA entities and their properties and relationships.

- ▶ Governance enablement profile extensions model

This model provides entities that are used as examples for the recommended approach for extending the default service level agreement (SLA) and service level definition (SLD) entities.

- ▶ Advanced Lifecycle Edition model

This model provides entities that represent the core entities in IBM Rational® Asset Manager to facilitate synchronization between Rational Asset Manager and WebSphere Service Registry and Repository.

- ▶ Manual endpoint model

The manual endpoint model allows you to create instances of different types of manual endpoint, where the location of the service that represents the endpoint can be specified as a URI.

For more information about governance enablement profile models, go to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_models.html

We describe these models in the sections that follow.

6.2.1 Service model

The service model provides entities that represent the derived entities that are created when WSDL, XSD, SCA module, and MQ artifacts are loaded. The derived entities include logical entities that represent elements of the WSDL

document and correlated service model entities that represent related service model entities. The service model is a part of both the basic profile and the governance enablement profile.

We provide a summary of the service model entities in this section.

Logical entities

Logical entities allow users to explore the content beyond the boundaries of the files or documents that are stored. They can have additional service description metadata allocated to them, such as properties, relationships, and classifications.

When a WSDL document is loaded into WSRR, the corresponding logical entities are created by the WSDL parser. For more information about the WSDL specification, refer to:

<http://www.w3.org/TR/wsd1>

The following key logical entities are created when WSDL documents are loaded:

- ▶ WSDLPortType
- ▶ WSDLBinding
- ▶ WSDLService
- ▶ WSDLPort
- ▶ WSDLOperation
- ▶ WSDLMessage
- ▶ SOAPAddress

The appropriate relationships between the logical entities and the source WSDL document are maintained. The logical entities are deleted automatically when the source document is deleted.

Service model entities

WSRR provides a correlator modifier plug-in that is enabled by default in the governance enablement profile. See 6.7.1, “The correlator modifier” on page 188 for details. It correlates WSDL logical objects to corresponding existing service model entities as a WSDL document is loaded and create new service model entities if there are no existing correlated entities. It maintains the relationship from service model entities to their corresponding derived logical entities. You can also create service model entities manually, before the corresponding WSDL, XSD, SCA module, or MQ artifacts are defined, and the modifier will carry out the required correlation when these artifacts are eventually defined and loaded.

In this section, we describe the key entities and the relationship between them, as illustrated in Figure 6-1.

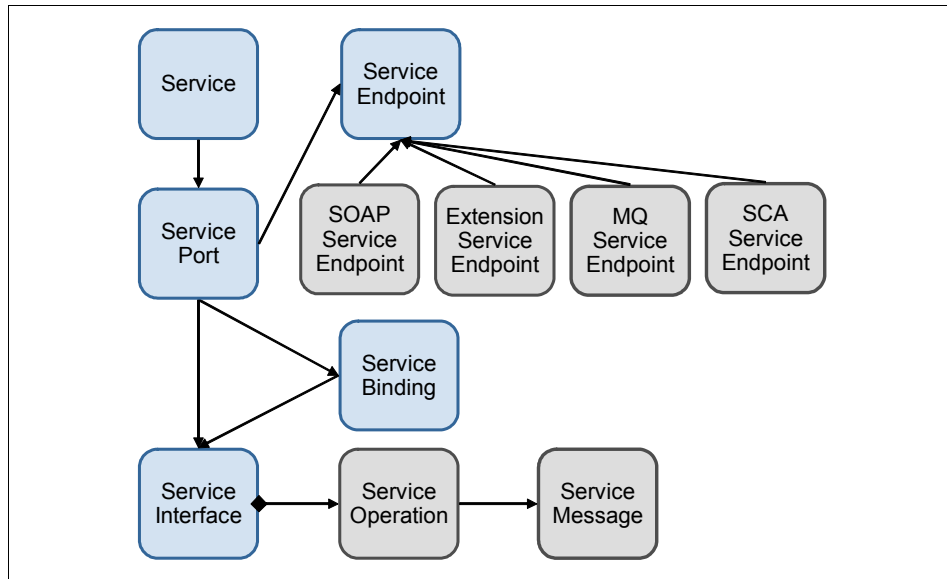


Figure 6-1 Key entities in the service model

For details of all the service model entities, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_models.html

Service entities

Service entities in the governance enablement profile represent a specific WSDL service that is defined by the service namespace, service name, and version. This service entity identifies and collects all service ports, and thus endpoints, that are available with that particular service version. A service entity is defined in the service model.

Figure 6-2 depicts the owned relationships that a service entity has with other entities in the governance enablement model.

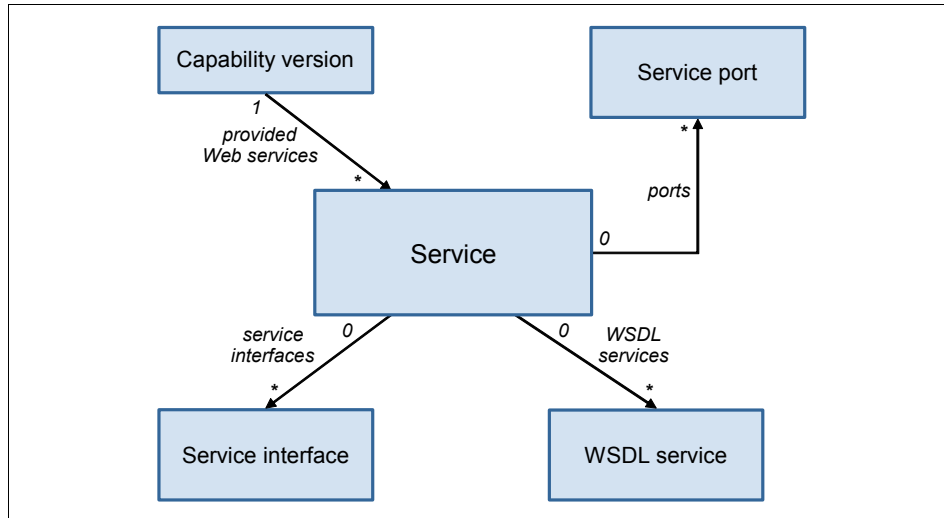


Figure 6-2 Service entity

Service port entity

A *service port* entity represents a specific WSDL port. It is defined by the port name and the name, namespace, and version of the service from which the port is derived. It contains all the endpoints that are associated with the port.

Service endpoint entity

A *service endpoint* entity in the governance enablement profile represents a distinct deployment of a named service port and provides the basic means of governing access to individual service endpoints. The following model types inherit their properties and relationships from this entity:

- ▶ SOAP service endpoint, which represents the governed service endpoint for any services that use SOAP addressing to define their endpoints
- ▶ MQ service endpoint, which represents the governed service endpoint for all types of MQ service

- ▶ SCA service endpoint, which represents the governed service endpoint for any non-SOAP services that are exposed from an SCA module.
- ▶ Extension service endpoint, which represents the governed service endpoint for any services that use WSDL extensions to define their endpoints

Service binding entity

A *service binding* entity represents a particular service interaction protocol, which is specified by a WSDL binding. It is defined by the binding namespace, binding name, and version.

Service interface entity

A *service interface* entity logically represents a particular service interface (corresponding to the WSDLPortType). It is defined by interface namespace, interface name, and version.

Service operation entity

A *service operation* entity represents a particular operation within a service interface. It is defined by the operation name and the name, namespace, and version of the service interface (WSDLPortType) from which the operation is derived.

Service message entity

A *service message* entity represents the service operation message. It is defined by its namespace, name, and version.

Extension service endpoint entity

An *extension service endpoint* entity in the governance enablement profile represents the governed service endpoint for any services that use WSDL extensions to define endpoints. The extension service endpoint entity has the same properties as a service endpoint entity and an optional relationship to an extension logical object in the service model.

MQ service endpoint entity

An *MQ service endpoint* entity in the governance enablement profile represents the governed service endpoint for all types of MQ services. An MQ service endpoint entity has the same properties as a service endpoint entity and an additional relationship to MQ endpoint in the service model.

SOAP service endpoint entity

A *SOAP service endpoint* entity represents the governed service endpoint for any services that use SOAP addressing to define their endpoints. A SOAP entity has

the same properties as a service endpoint and additional relationships to a manual SOAP endpoint or SOAP address and the WSDL port.

SCA service endpoint

An *SCA service endpoint* represents the governed service endpoint for any non-SOAP services that are exposed from an SCA module. SOAP1.1 exports are represented by a SOAP service endpoint, and SOAP1.2 exports are represented by an extension service endpoint.

An SCA service endpoint has the same properties as a service endpoint, plus the relationships to the SCA export binding and WSDLPortType.

6.2.2 Governance enablement model

The governance enablement model provides the entities, together with relationships and properties that are defined on those entities, to enable an SOA governance solution within the enterprise. Business users can identify business capabilities that are required by the organization. Development users, in conjunction with SOA governance and operations users, can collaborate and define the contracts between departments by defining documents of understanding (DOUs), service level agreements (SLAs), and service level definitions (SLDs).

Figure 6-3 illustrates the entities in the governance enablement model.

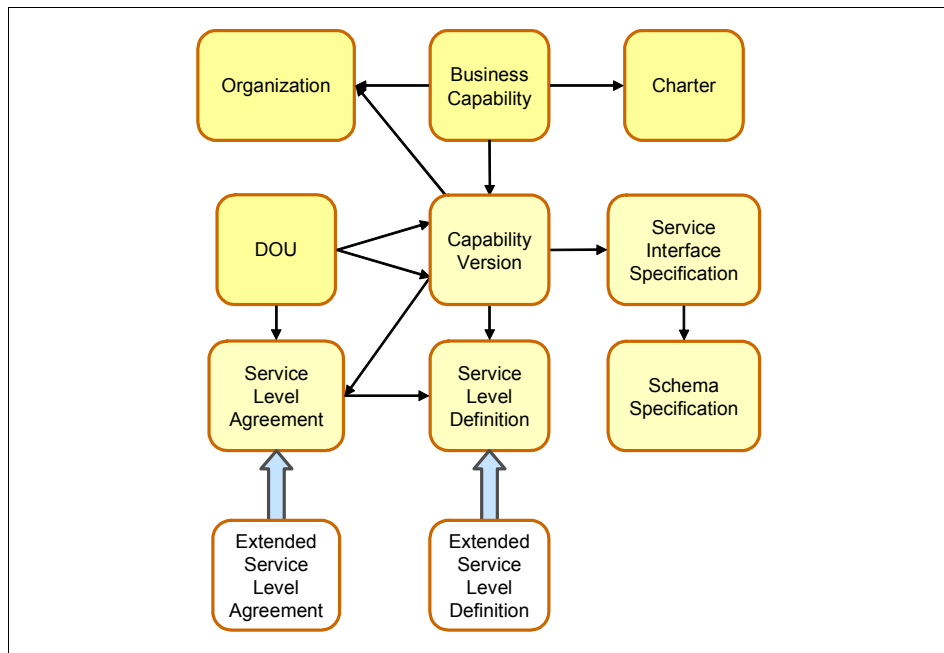


Figure 6-3 Entities in the governance enablement profile model

This section provides an overview of the entities in the governance enablement model, their properties, and their relationships to the service model entities.

Business capability entity

A *business capability* entity represents a construct that expresses a generalized (unrealized) feature within the SOA environment. This entity is used as the starting point for linking business requirements with technical functionality in the SOA environment.

The following model types inherit their properties and relationships from this entity:

- ▶ Business application - An existing application or a particular channel to market, such as a web or portal application
- ▶ Business process - A collection of related activities or tasks in an organization
- ▶ Business service - An unrealized service in the organization

Figure 6-4 depicts the owning and reverse relationships that a business capability entity has with other entities in the governance enablement profile model.

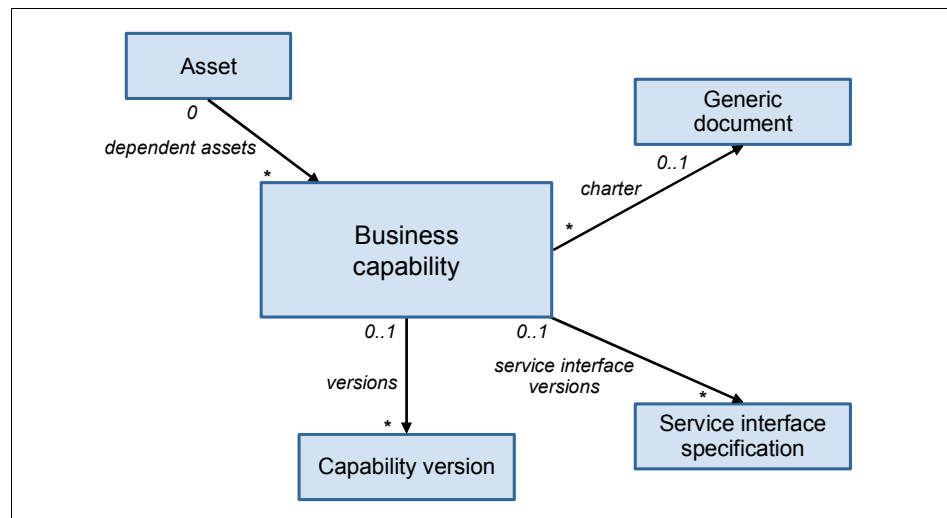


Figure 6-4 Business capability entity

This entity inherits properties and relationships from the asset entity plus its own additional properties as defined in the governance enablement profile model.

For a detail description of the properties and relationships for the business capability entity, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_business_capability.html

Capability version entity

A *capability version* entity defines a specific version of a feature to realize specific versions of the business capability entities (application, process, and service). This entity also specifies the following main characteristics for a particular version:

- ▶ A set of SLDs that are the formal specification for any provided services, including both functional and quality of service (QoS) characteristics
- ▶ A set of SLAs that are the agreed specification for any consumed services and reference the particular SLD that must be used for any interactions
- ▶ A definition of the SCA modules and web services that deliver the capability in this particular version

The following model types inherit properties and relationships from this entity:

- Application version

A specific version or release of a web application that consumes only services
- Process version

A specific version or release of an SOA process that can expose various of its capabilities as services with appropriate SLDs
- Service version

A specific version or release of a business service that provides a range of functional and non-functional specifications

Figure 6-5 depicts the owning and reverse relationships that a capability version entity has with other entities in the governance enablement model.

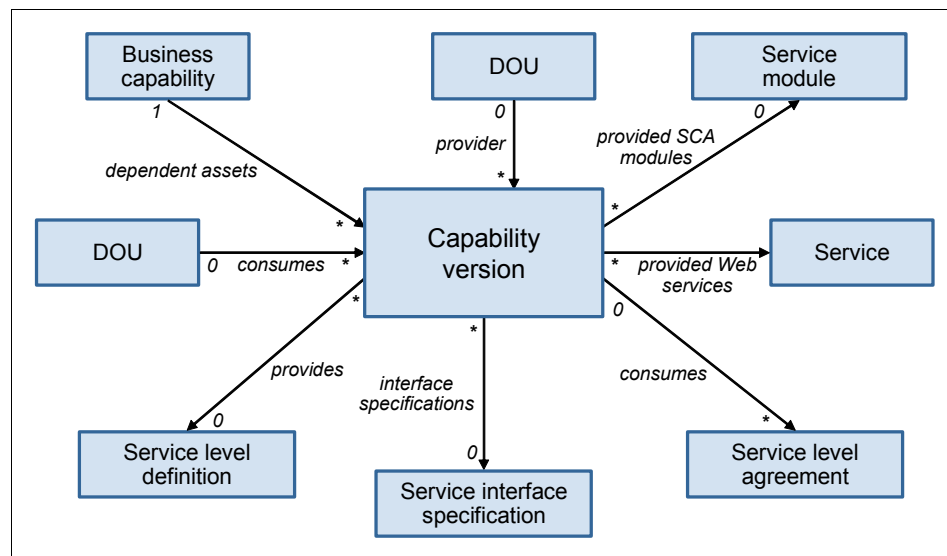


Figure 6-5 Capability version entity

This entity inherits properties and relationships from the business capability entity plus the additional properties as defined in the governance enablement model.

One of the properties of a capability version is *consumer identifier* (`gep63_consumerIdentifier`), which is used to identify the consuming capability version in any interaction. A capability version can be a consumer, a provider, or both a consumer *and* a provider. During any interactions between the consumer and provider, if the consumer identifier is included in the interaction header, the

provider ESB can identify the particular capability version that invokes the endpoint.

A capability version owns relationships to the service module or service that provides its capability version, to the service interface specification that it uses, to the SLDs that it supports, and to all SLAs that it consumes to provide its capability version.

SCA modules declare the endpoints that they import (*imports*) and the endpoints that they provide (*exports*). This relationship allows the capability version entity to be mapped to the provided and consumed endpoints from the SCA module. The SCA module forms part of the SCA description of the service or process and is expressed in Service Component Definition Language (SCDL) during the development of the capability.

The *consumes* relationship defines the endpoints and the QoS for each SLA entity that is used in any interaction with the provider.

The *provides* relationship defines the formal definition of the interaction and nonfunctional characteristics for any interaction, including the physical communication mechanisms that are used to deliver the messages for interaction with the provided services and the QoS characteristics that are related to the interaction, such as security and identity.

The *provided web services* relationship corresponds to the specific versions of services that are declared in WSDL as part of the capability version (service version, process version, and application version) entity. From the service, consumers can navigate to the bindings and interfaces that are supported for this particular web service.

For a detail description of the properties and relationships for the capability version, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_capability_version.html

Document of understanding entity

A *document of understanding* (DOU) entity represents an agreement between organizations that details how the consuming organization's realization of their capability (capability version entity) uses the providing organization's realized capability.

Figure 6-6 depicts the owning relationships that a DOU entity has with other entities in the governance enablement model.

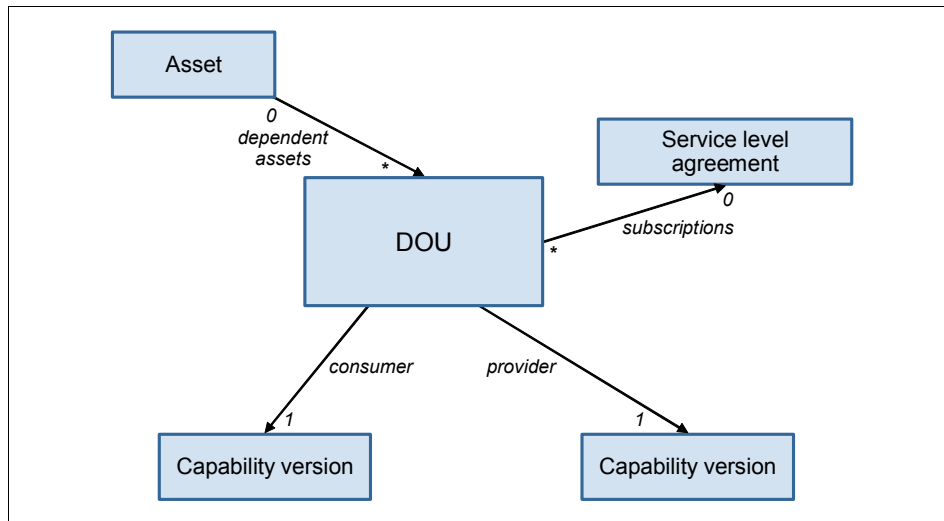


Figure 6-6 DOU entity

A DOU entity inherits properties and relationships from the asset entity and from the additional relationships to the SLAs that realize it, the consumer capability version and the provider capability version.

Schema specification entity

A *schema specification* entity defines shared schema documents, which allow the schema specification entities to be governed independently. Figure 6-7 depicts the owning relationships that a schema specification entity has with other entities in the governance enablement model.

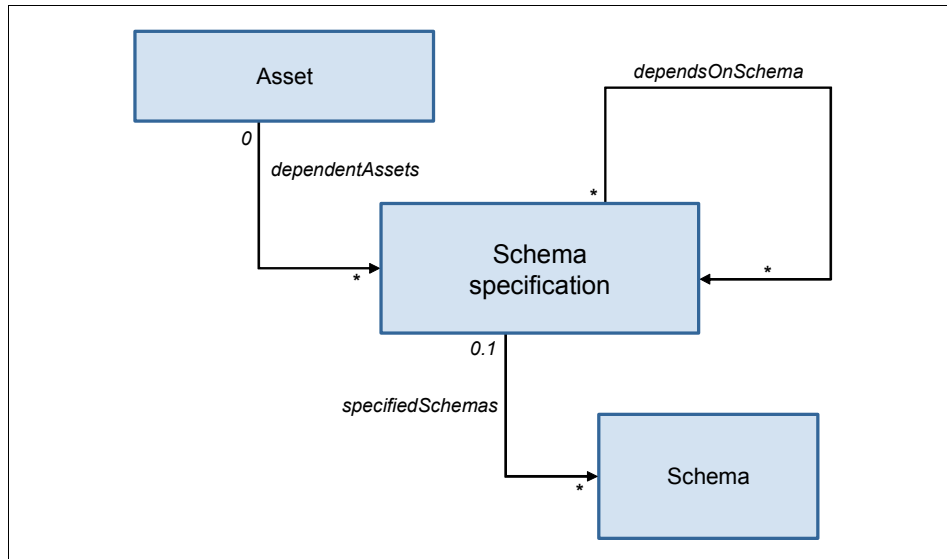


Figure 6-7 Schema specification model

A schema specification entity has the same properties and relationships as an asset entity, the additional owned relationships to the schemas that it defines, and other schema specifications on which it depends.

Service interface specification entity

A *service interface specification* entity defines a particular interaction pattern and message structure that is supported throughout the realized versions of business capabilities. This entity is independent of any transport or QoS. The interface specification usually refers to a set of WSDL port types that are shared throughout the capability versions.

Interface version numbers have the format *major.minor*. A single major interface version represents a backward-compatible set of minor interface versions, where each minor version extends only the existing interface and makes no changes to interfaces or operations that are declared in earlier minor versions.

Figure 6-8 depicts the owning and reverse relationships that a service interface specification entity has with other entities in the governance enablement model.

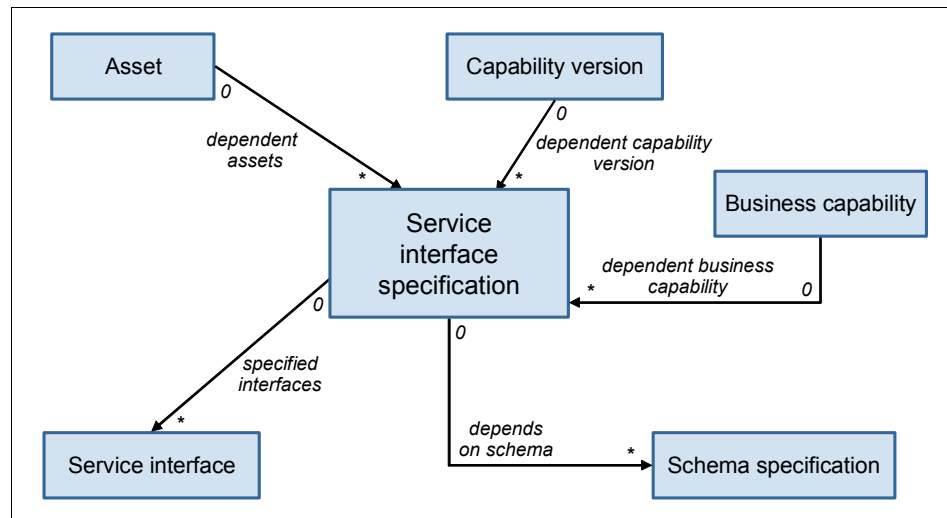


Figure 6-8 Service interface specification entity

A service interface specification entity has the same properties and relationships as an asset entity and the additional owned relationships to the schema specifications on which it depends and service interfaces that it defines.

Service level agreement entity

A *service level agreement* (SLA) entity defines a specific dependency that a capability version entity (for example a service version, process version, or application version) has on a particular SLD that is provided by another service version. Although the SLA specifies a particular SLD, the implication is that backward-compatible SLDs can be substituted and will support the same SLAs.

Figure 6-9 depicts the owned and reverse relationships that an SLA entity has with other entities in the governance enablement model.

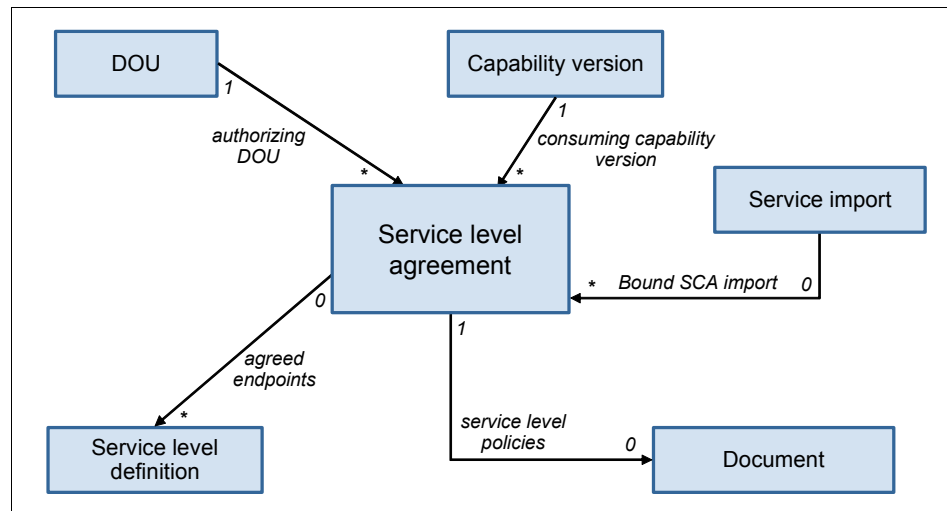


Figure 6-9 SLA entity

For a detailed description of the properties and relationships for the SLA entity, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_service_level_agreement.html

Service level definition entity

The *service level definition* (SLD) entity provides a formal specification of the physical communication mechanisms that are used to deliver the messages for interaction with a provided service. This entity includes non-functional characteristics that are related to the interaction, such as security and identity. Examples of an SLD include:

- ▶ For a web service, the WSDL port (with various policy assertions using WS-Policy syntax), the binding (and policies), and the relationship to the service interface (port type)
- ▶ For an SCA export, the export (with various policy assertions using WS-Policy syntax), the SCA binding that is defined in the export, and the relationship to the service interface

In either case, a list of callable and interchangeable endpoints that support the SLD entity is provided.

Figure 6-10 depicts the owned and reverse relationships that an SLD entity has with other entities in the governance enablement model.

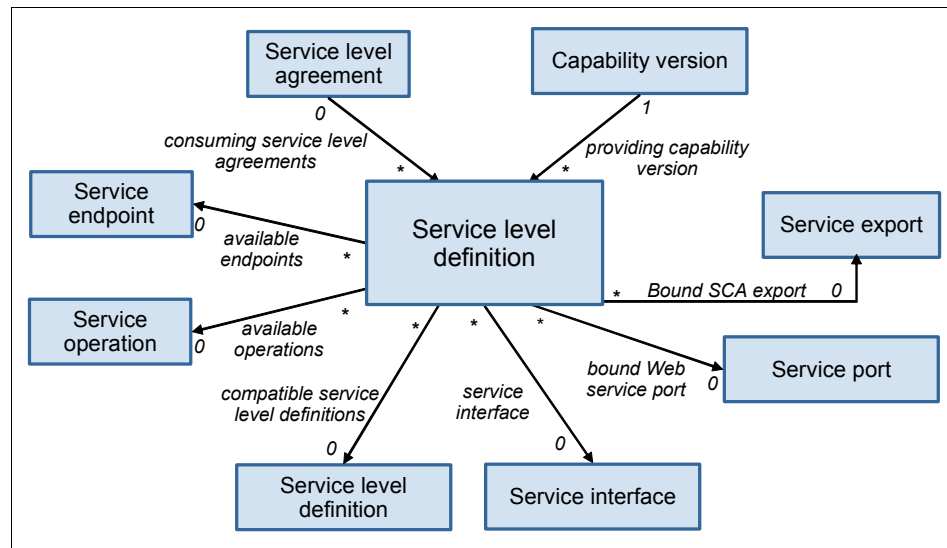


Figure 6-10 SLD entity

For a detailed description of the properties and relationships for the SLD entity, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_governance_enablement_model.html

6.2.3 Governance enablement profile extensions model

The governance enablement profile extensions model provides entities that extend the SLA and SLD entities defined in the governance enablement model. These extensions are examples of the suggested approach to extend the default SLA and SLD entities provided in the governance enablement profile.

Extended SLA entity

The *Extended SLA* entity represents the details of a specific service subscription and defines the QoS properties that are used in any interactions between the consuming capability version and the agreed provider endpoints. An Extended

SLA entity has the same properties and relationships as an SLA entity and the additional properties that are defined by the governance enablement profile extension model. For a detailed description of its extended properties, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_extended_sla.html

Extended SLD entity

The *Extended SLD* entity provides additional QoS information for a subscribable endpoint. The Extended SLD entity can be used to define particular metrics that are associated with endpoints. An Extended SLD entity has the same properties and relationships as an SLD entity, and additional properties as defined in the extension model. For a detailed description of its extended properties, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_extended_sld.html

6.2.4 Manual endpoint model

The manual endpoint model allows you to create instances of different types of manual endpoint, where the location of the service that represents the endpoint can be specified as a URI. Also, you can associate a WSDLPortType with the endpoint to describe the interface that the endpoint supports. These objects are not correlated by the correlator modifier.

The model includes the following manual endpoints:

- ▶ Interfaced manual endpoint
This endpoint is the abstract parent of different types of interfaced manual endpoints. This endpoint cannot be instantiated.
- ▶ JMS endpoint
The JMS endpoint extends the interfaced manual endpoint.
- ▶ MQ endpoint
The MQ endpoint extends the interfaced manual endpoint.
- ▶ HTTP endpoint
The HTTP endpoint extends the interfaced manual endpoint. Its properties are inherited from the parent object.

6.2.5 Advanced Lifecycle Edition model

The Advanced Lifecycle Edition model provides the core asset and organization entities that are represented in IBM Rational Asset Manager. An asset represents an entity that can be synchronized between Rational Asset Manager and WSRR. Therefore, those entities that are subclasses of the asset entity, including those in the governance enablement model, can be synchronized between Rational Asset Manager and WSRR.

Asset entity

An *asset* entity represents any object type in WSRR that might correspond to an asset in Rational Asset Manager or other federated repository. If a WSRR entity is defined as an asset, that entity can also use Rational Asset Manager or WSRR Advanced Lifecycle Edition capabilities to govern that asset.

An asset entity is found in the Advanced Lifecycle Edition model. Figure 6-11 depicts the owning and reverse relationships that an asset entity has with other entities in the governance enablement profile.

Dependency relationship: The dependency relationship is used only if the profile does not provide a more relevant relationship to represent the dependency.

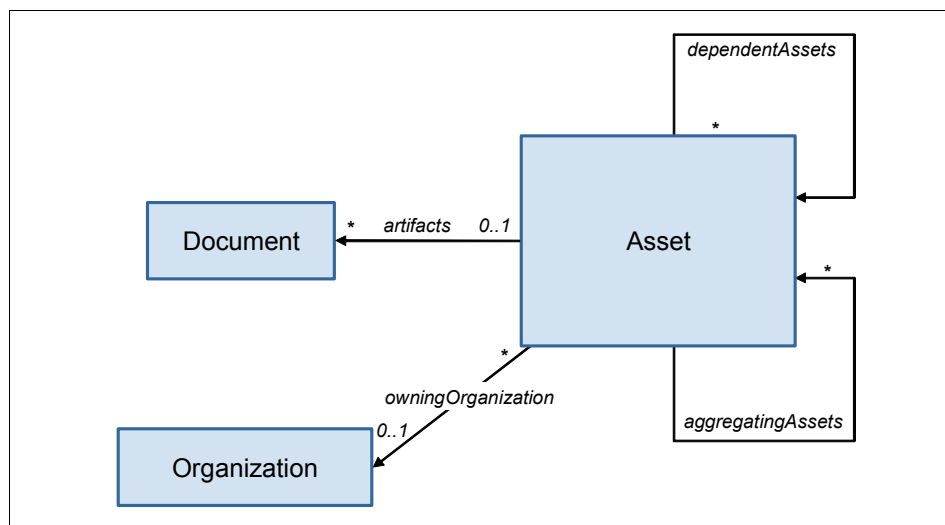


Figure 6-11 Asset entity relationship

Table 6-1 describes the properties that are defined for an asset entity from the Advanced Lifecycle Edition model.

Table 6-1 Asset entity properties

Displayed property name	Actual property name	Description	Type
Name	name	A required descriptive name for the entity	String
Description	description	A textual description of the entity	String
Namespace	namespace	The XML schema namespace for the entity	String
Version	version	The version of the entity	String
GUID	ale63_guid	The globally unique identifier used to identify the asset in Rational Asset Manager	String
Full description	ale63_fullDescription	A full textual description of the entity held in Rational Asset Manager	String
Asset type	ale63_assetType	The specific asset type used in Rational Asset Manager to determine the type, and therefore the properties and relationships of the entity	String
Asset web link	ale63_assetWebLink	A URL that, if followed, opens a browser window in Rational Asset Manager showing the details of this particular entity	String
Remote state	ale63_remoteState	The state of the asset in Rational Asset Manager	String
Asset owners	ale63_assetOwners	Those users that have ownership permissions or roles in Rational Asset Manager	String
Owner email	ale63_ownerEmail	A default email address to use for all inquiries or notifications about this Asset entity	String
Community name	ale63_communityName	The community to which this asset belongs in Rational Asset Manager	String

The business capability, capability version, DOU, schema specification, and service interface specification entities inherit properties and relationships from the asset entity.

Organization

An *organization* is used to group assets that have common stakeholder roles and that require high degrees of collaboration. You can use separate organizations to represent organizational divisions or role divisions (governance, development, and operations). Assets are, however, owned only by one organization.

6.2.6 Governance enablement model and service model relationships

Governance enablement model entities are related to service model entities to provide a complete SOA governance model. This SOA governance model relates business concepts (business capabilities, and organizations) and governance concepts (SLA and SLD) to the service model entities that represent the IT view of services. This relationship facilitates implementing governance processes and controls to align business and IT goals.

Figure 6-12 gives an overview of the relationships between the two models.

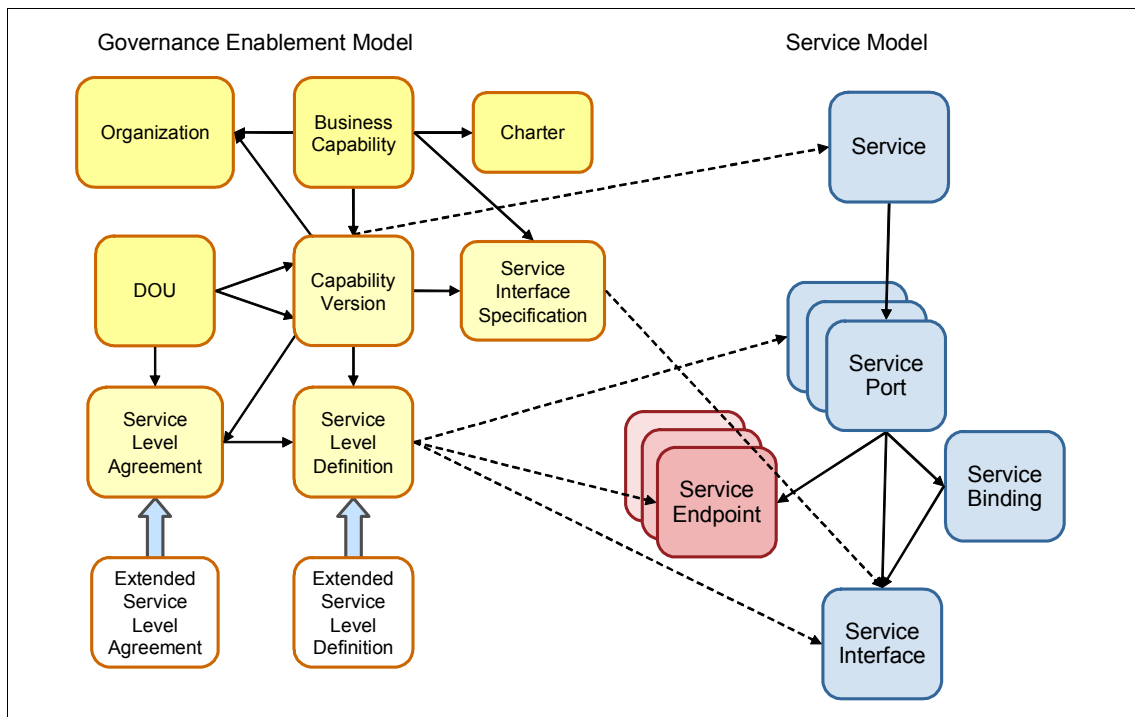


Figure 6-12 Relationships between the governance enablement model and the service model

Service model objects are created appropriately as needed by the correlator modifier when a WSDL document is loaded. For more information about the correlator modifier, see 6.7.1, “The correlator modifier” on page 188. Business model objects are typically created manually.

6.3 Roles and access control

The governance enablement profile provides the following roles:

- ▶ The *Business* role defines the governance processes, policies, and standards that are shared across the enterprise to ensure effective interoperability, agility, and robustness of the SOA solutions. In an organization, the Business role represents business managers, analysts, and subject matter experts who are interested in how the SOA services and processes contribute to the business.
- ▶ The *SOA Governance* role represents architects, architecture boards, and SOA centers of excellence. This role also includes individuals from other roles (business, development, and operations) who define the governance processes, policies, and standards that are shared throughout the organization to ensure effective interoperability, agility, and robustness of SOA solutions.
- ▶ The *Development* role represents software development practitioners, including architects, release managers, software developers, testers, assembly developers, integrators, and asset librarians. They develop the software specifications and implementations to realize the requirements that are provided by the business and ensure that the implementation meets the business needs and adheres to the governance standards. Development roles usually are associated with a specific organization or department, with responsibility for delivering implementations to support a particular area of the business.
- ▶ The *Operations* role represents operations managers, operations architects, system administrators, integration testers, and IT resource managers. They manage the IT infrastructure, and deploy, configure, and test the implementations produced by development. They are responsible for operational QoS and capacity planning, and although their main activities are later in the life cycle, they are involved in all specification activities and reviews to ensure that the planned capabilities can be successfully delivered. Operations roles can be centralized or arranged by line of business or organization, according to individual company preferences.

Business users can identify capabilities that are required by the organization. Development users, in conjunction with SOA governance and operations users,

can collaborate and define the contracts between departments by defining DOUs, SLAs, and SLDs. These defined entities can eventually be hooked in to implementation entities when the original business capabilities are realized in to runnable services and represented in the profile.

Figure 6-13 illustrates the relationships between the governance enablement model entities and the key service model entities. It also indicates the governance enablement profile roles that are responsible for these governance model entities.

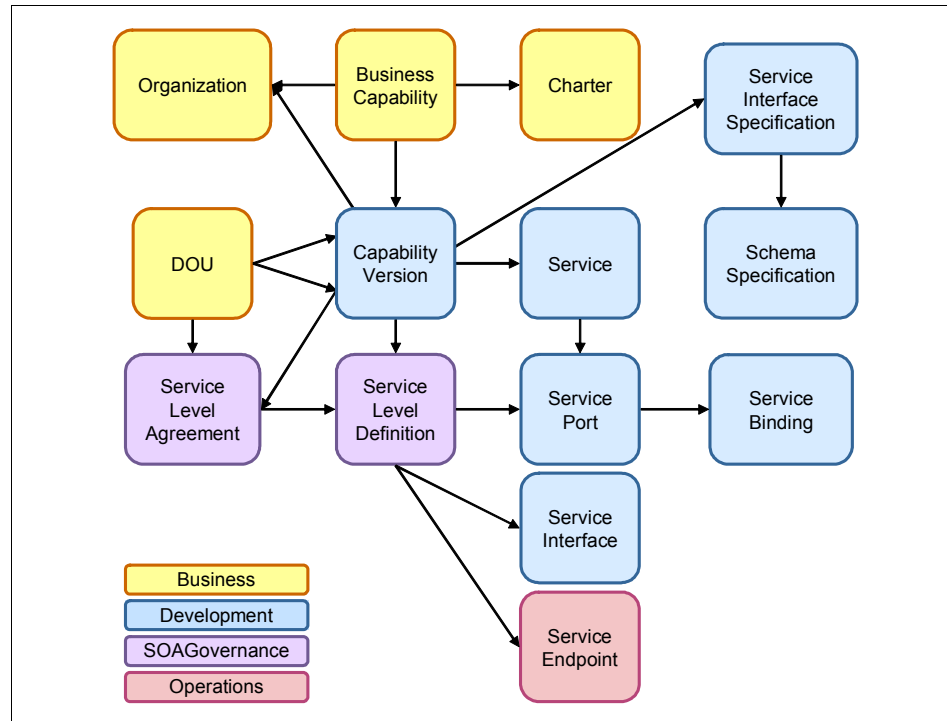


Figure 6-13 Access roles for governance model entities

6.4 Life cycles in the governance enablement profile

Entities from the various models in the governance enablement profile are governed by moving them through states in a *life cycle*. The life cycle state of an entity indicates its progress in the development process. Table 6-2 describes the life cycles that are defined in the governance enablement profile and the entities that pass through them.

Table 6-2 *Life cycles and their associated entities*

Name	Usage	Governed entity
Asset life cycle	Govern an entity from initial identification to retired	DOU, service interface specification, and schema specification
Capability life cycle	Govern a capability from initial identification to retired	Business capability, business application, business process, and business service
SOA life cycle	Govern a capability version from initial identification to retired	Capability version, application version, process version, and service version
SLD life cycle	Govern an SLD from initial identification to retired	SLD
SLA life cycle	Govern an SLA from initial identification to retired	SLA
Endpoint life cycle	Govern an endpoint from being approved for use to retired	Service endpoint, MQ service endpoint, SOAP service endpoint, and extension service endpoint

The next sections describe each of the governance enablement profile life cycles.

6.4.1 The asset life cycle

The asset life cycle in the governance enablement profile is used to govern an asset entity from being initially identified, through to being approved for reuse by consumers, and eventually to being withdrawn from use. Figure 6-14 depicts the transitions and states of an asset entity as defined in the asset life cycle.

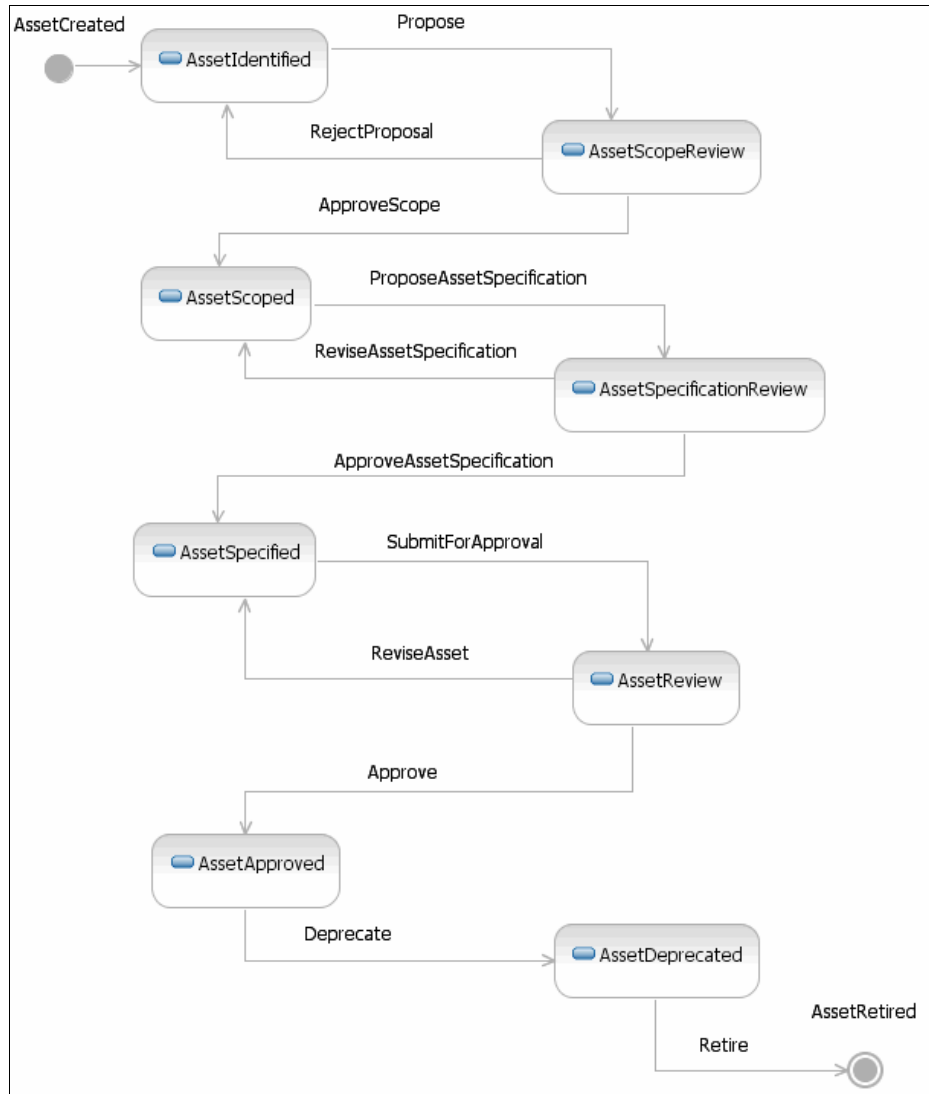


Figure 6-14 Asset life cycle

Table 6-3 describes the states of the asset life cycle and, for each state, names the transition that moves an asset entity forward to that state.

Transition note: Where there is a transition that moves an asset entity from one state back to a previous state, that transition is not listed in Table 6-3. Refer to Figure 6-14 to see all transitions.

Table 6-3 Asset entity life cycle explanation

Transition	State	Description
Initial state	Asset identified	The requirements for an asset are defined. The scope for its use, and the other SOA artifacts that will depend on this asset, are also being made clear so that the purpose of the asset can be readily identified.
Propose	Asset scope review	The requirements proposed for the asset are considered and a decision is made as to whether further development of the asset is to be undertaken. If an asset proposal is rejected, it can be proposed again after modification.
Approve proposal	Asset scoped	The main development work on the asset starts, with a specification being produced according to the agreed scope and requirements. When the development team decides that the specification is complete, it is submitted for review.
Propose asset specification	Asset specification review	The specification for the asset is reviewed. Specifications that are rejected can be proposed again after modification.
Approve asset specification	Asset specified	The main asset development work is done according to the specification.
Submit asset for approval	Asset review	The asset is reviewed by the stakeholders for conformance with the requirements that were approved. If these requirements are deemed not to be met, the asset is rejected and goes back to development for rework.
Approve	Asset approved	The asset is visible to potential consumers for reuse.
Deprecate asset	Asset deprecated	The asset is not available to new users but is still be visible to existing subscribers and consumers.
Retire asset	Asset retired	The asset has been withdrawn from use and there are no active consumers using this asset. It ceases to be visible to general users.

6.4.2 The capability life cycle

The capability life cycle in the governance enablement profile is used to govern a business capability entity from being initially identified, through to being approved when all the governance requirements are satisfied, and eventually to being deprecated and retired when it is no longer necessary. Figure 6-15 depicts the transitions and states of a business capability entity as defined in the capability life cycle.

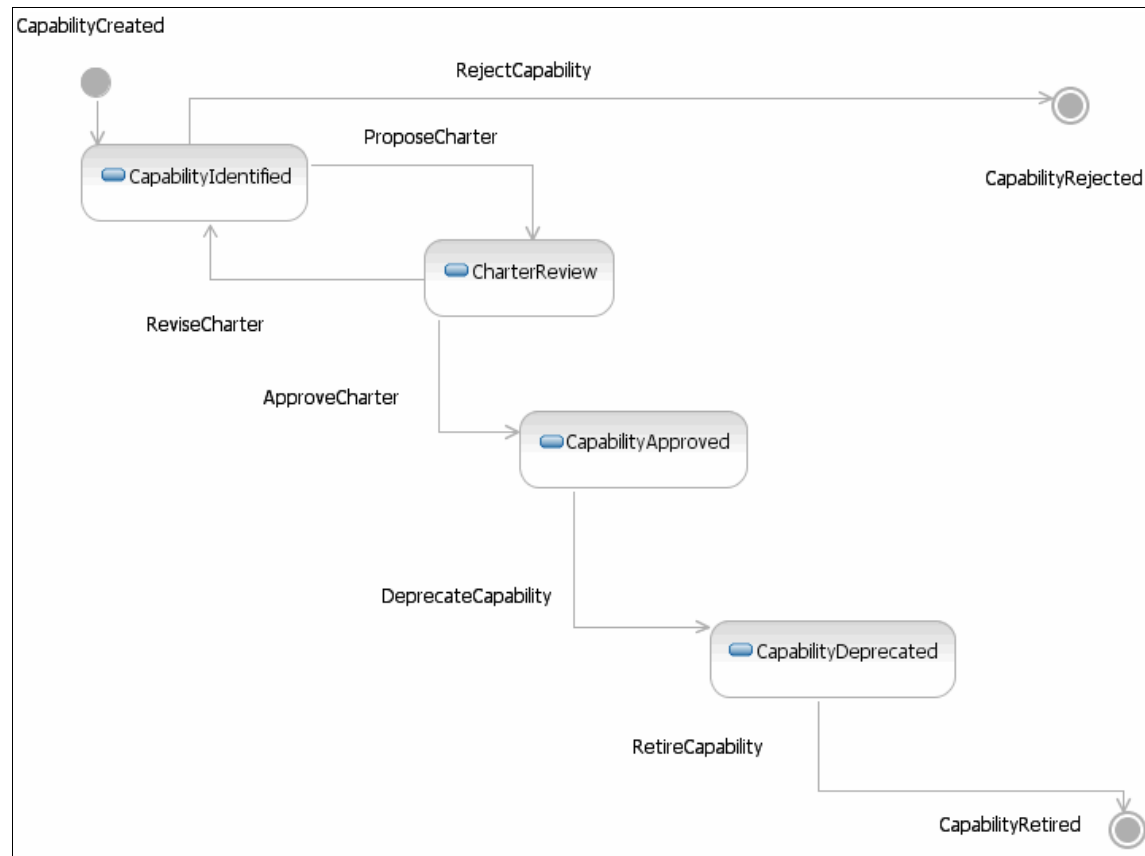


Figure 6-15 Capability life cycle

Table 6-4 describes the states of the capability life cycle, and, for each state, names the transition that moves a business capability entity forward to that state.

Transition note: Where there is a transition that moves a business capability entity from one state back to a previous state, that transition is not listed in Table 6-4. Refer to Figure 6-15 to see all transitions.

Table 6-4 Capability life cycle explanation

Transition	State	Description
Initial state	Capability identified	This state is entered when a business capability is first identified. During this state, a charter is produced to scope the capability that is required and agree where in the organization the responsibility for delivering this capability is to be assigned.
Reject capability	Capability rejected	Charter review has determined the capability is not needed.
Propose charter	Charter review	Either approved or sent back for revision.
Approve charter	Capability approved	Governance requirements for the business capability are met.
Deprecate capability	Capability deprecated	Business capability is still available for existing users but has been superseded.
Retire capability	Capability retired	Business capability versions are not longer in use.

6.4.3 The SOA life cycle

The SOA life cycle in the governance enablement profile is used to govern a capability version entity from being initially identified, through to being deployed in production, and eventually deprecated when it is no longer required. The SOA life cycle is separated into multiple phases, which we describe in the following sections.

Model phase

The model phase of the SOA life cycle in the governance enablement profile is used to govern a capability version entity from being initially identified, through to its specification being proposed for review. Figure 6-16 depicts the transitions and states of the model phase of an entity as defined in the SOA life cycle.

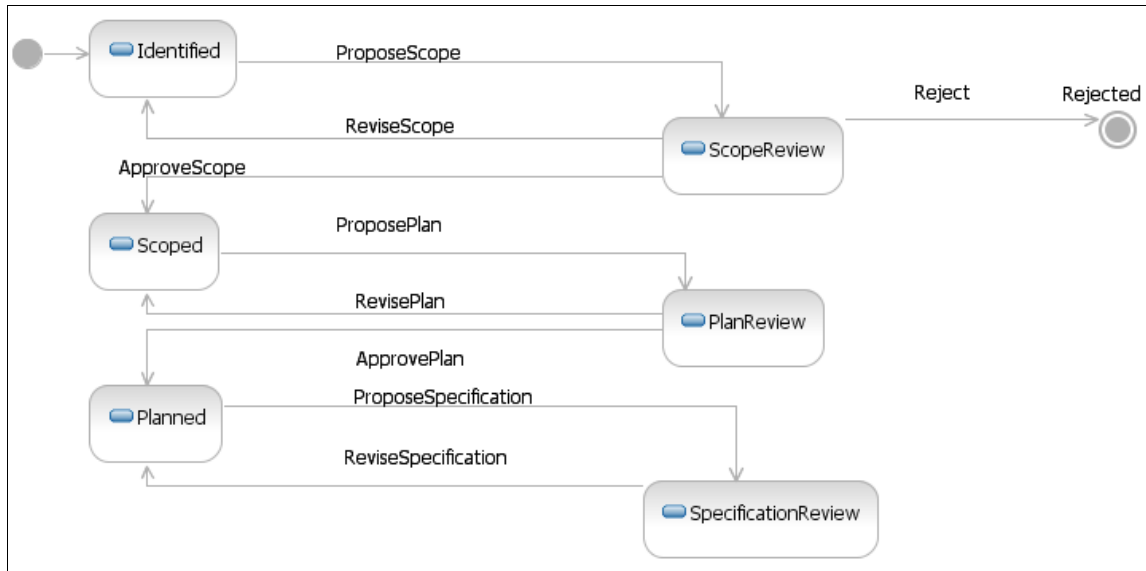


Figure 6-16 SOA life cycle, model phase

Table 6-5 describes the states of the model phase of the SOA life cycle and, for each state, names the transition that moves an entity forward to that state.

Transition note: Where there is a transition that moves an entity from one state back to a previous state, that transition is not listed in Table 6-5. Refer to Figure 6-16 to see all transitions.

Table 6-5 SOA life cycle model phase explanation

Transition	State	Description
Initial state	Identified	A new version of a capability has been requested or identified. The stakeholders identify the requirements for this version of the capability.
Propose scope	Scope review	The stakeholders review the requirements and intended ownership of the version, and agree the scope. New stakeholders and requirements might be identified, requiring the scope to be revised and proposed again. Any new version of a capability must be mapped to an approved business capability to ensure that its role in the organization is clear.
Approve scope	Scoped	The development and owning organizations must work together to define the funding and time frames for the project.
Propose plan	Plan review	The various lines of business involved in the provision and consumption of the capability now have the opportunity to review the details of the implementation of the capability version.
Approve plan	Planned	Development of the capability begins. The first activities involve agreeing the specification for the version. This includes the complete definition of all exposed interfaces, and any provided SLDs that other capabilities can use, together with any SLAs for capabilities that this version will consume. Development of these specifications is likely to require development of the actual implementations to ensure that the specification can be realized.
Propose specification	Specification review	The specification review must ensure that the specifications that are provided for this version of the capability will meet the stakeholder requirements. Any use of this version by other consumers, as defined by the SLDs, will be based entirely on the specification and costed accordingly. Any dependencies on other capability versions must be specified and agreed through an SLA and DOU. Any disagreement about functional specifications might require the specification to be revised and proposed again. Approval of the version specification determines a contract that allows both potential consumers and providers to proceed independently.

Assemble phase

The *assemble* phase of the SOA life cycle in the governance enablement profile is used to govern a Capability version entity from having its specification approved, through to being realized. Figure 6-17 depicts the transitions and states of the assemble phase of an entity as defined in the SOA life cycle.

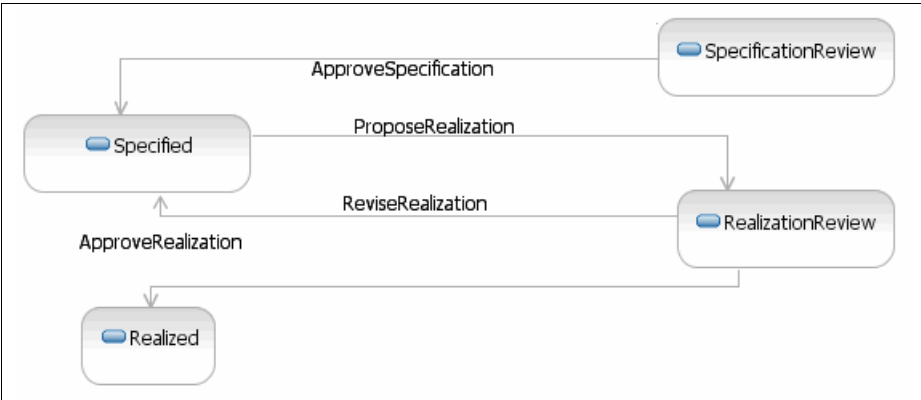


Figure 6-17 SOA life cycle assemble phase

Table 6-6 describes the states of the assemble phase of the SOA life cycle and, for each state, names the transition that moves an entity forward to that state.

Transition note: Where there is a transition that moves an entity from one state back to a previous state, that transition is not listed in Table 6-6. Refer to Figure 6-17 to see all transitions.

Table 6-6 SOA life cycle assemble phase explanation

Transition	State	Description
Approve specification	Specified	In this state, the majority of the development or assembly of this version of the capability occurs. The details of activities undertaken during this state will be specific to the development processes being used within the organization. but the key exit criteria is when development (and development test) is complete and a release is ready to be reviewed before being sent to operations for integration testing, configuration and staging. On completion of the development, the version realization is proposed for a realization review.

Transition	State	Description
Propose realization	Realization review	In this state, stakeholders agree that sufficient testing of the developed realization has taken place, and the quality of the version is such that it can be sent to operations for deployment and configuration into the staging environments.
Approve realization	Realized	The version is installed onto staging (integration, test, pre-production) environments and configured to operate with other deployed applications, processes and services. Testing is undertaken of the SLDs that the capability offers. and if all SLDs can be met then the version is proposed as ready for staging.

Deploy phase

The *deploy* phase of the SOA life cycle in the governance enablement profile is used to govern a capability version entity from being realized, through to being deployed in production. Figure 6-18 depicts the transitions and states of the deploy phase of an entity as defined in the SOA life cycle.

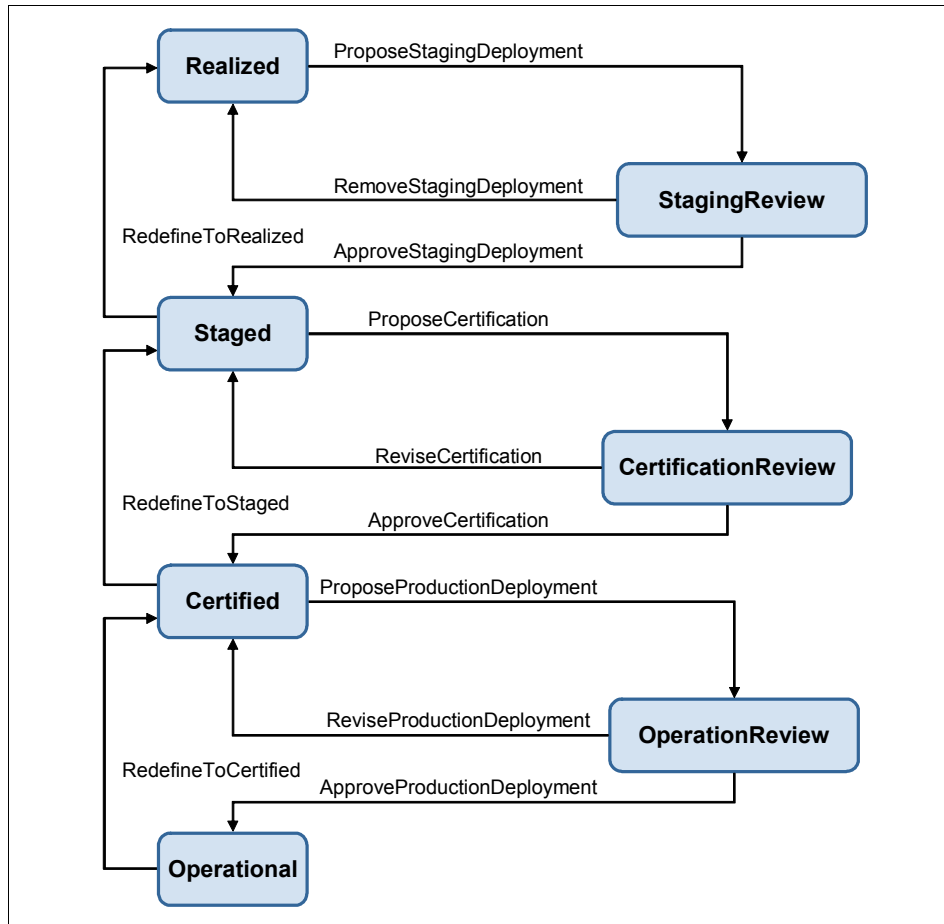


Figure 6-18 SOA life cycle deploy phase

Table 6-7 describes the states of the deploy phase of the SOA life cycle, and, for each state, names the transition that moves an entity forward to that state.

Transition note: Where there is a transition that moves an entity from one state back to a previous state, that transition is not listed in Table 6-7. Refer to Figure 6-18 to see all transitions.

Table 6-7 SOA life cycle deploy phase explanation

Transition	State	Description
Propose staging deployment	Staging review	This state identifies when consumers in the staging environment can have access to the SLDs that have been deployed. The review must review the SLAs expected of the capability to ensure that all planned commitments can be met. Any capacity deficit can result in a revision of the staging deployment and configuration, requiring a reproposal to review.
Approve staging deployment	Staged	The SLAs and dependencies between capabilities are tested to prove that when this loosely coupled solution is deployed in an operational environment, most of the changes that deliver business agility have been tested over the likely bounds of use. When this testing is complete, the version is proposed for certification.
Redefine to realized	Staged	The Staged state was previously reached but now needs to be backed out into a Realized state.
Propose certification	Certification review	The staging and integration testing is confirmed and authority is given to deploy the version to the production environment.
Approve certification	Certified	The certified version and configuration is deployed into the production environments and tested.
Redefine to staged	Certified	The Certified state has previously been reached, but now needs to be backed out into a Staged state.
Propose production deployment	Operational review	The final confirmation and authority is given to release the version of the capability to the business.
Approve production deployment	Operational	The capability version is deployed to production.
Redefine to certified	Operational	The Operational state was previously reached but now needs to be backed out into a Certified state.

Manage phase

The *manage* phase of the SOA life cycle in the governance enablement profile is used to deprecate, and eventually retire, a deployed Capability version entity when it is no longer required. Figure 6-19 depicts the transitions and states of the manage phase of an entity as defined in the SOA life cycle.

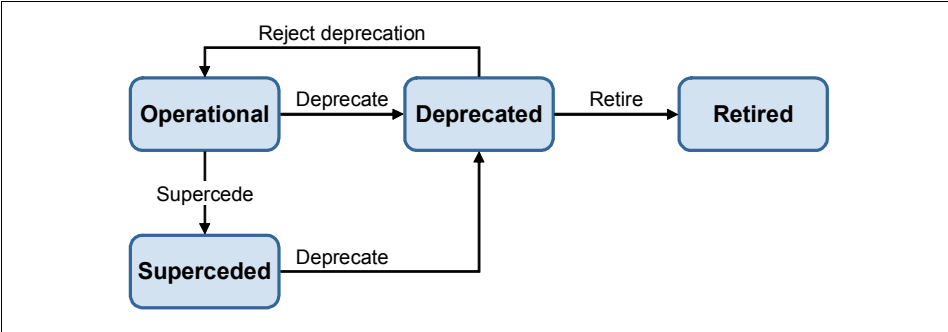


Figure 6-19 SOA life cycle manage phase

Table 6-8 describes the states of the manage phase of the SOA life cycle and, for each state, names the transition that moves an entity forward to that state.

Transition note: Where there is a transition that moves an entity from one state back to a previous state, that transition is not listed in Table 6-8. Refer to Figure 6-19 to see all transitions.

Table 6-8 SOA life cycle manage phase explanation

Transition	State	Description
Supercede	Superceded	The capability version, application version, process version, or service version is replaced by a more recent version. It will be deprecated at some point in the future, so that consumers have time to move to the more recent version.
Deprecate	Deprecated	The business capability is no longer considered necessary. No more versions of this capability must be produced but use of existing versions can continue until no longer required.
Retire	Retired	There are no versions of this capability in use in the organization.

6.4.4 The SLD life cycle

The SLD life cycle in the governance enablement profile is used to govern an SLD from being initially identified, through to being retired when it is no longer in use. The SLD life cycle is separated into multiple phases, which we describe in the following sections.

Model and assemble phases

The *model* and *assemble* phases of the SLD life cycle are used to govern an SLD entity from being initially identified through to having its specification approved. Figure 6-20 depicts the transitions and states of the model and assemble phases of an SLD entity as defined in the SLD life cycle.

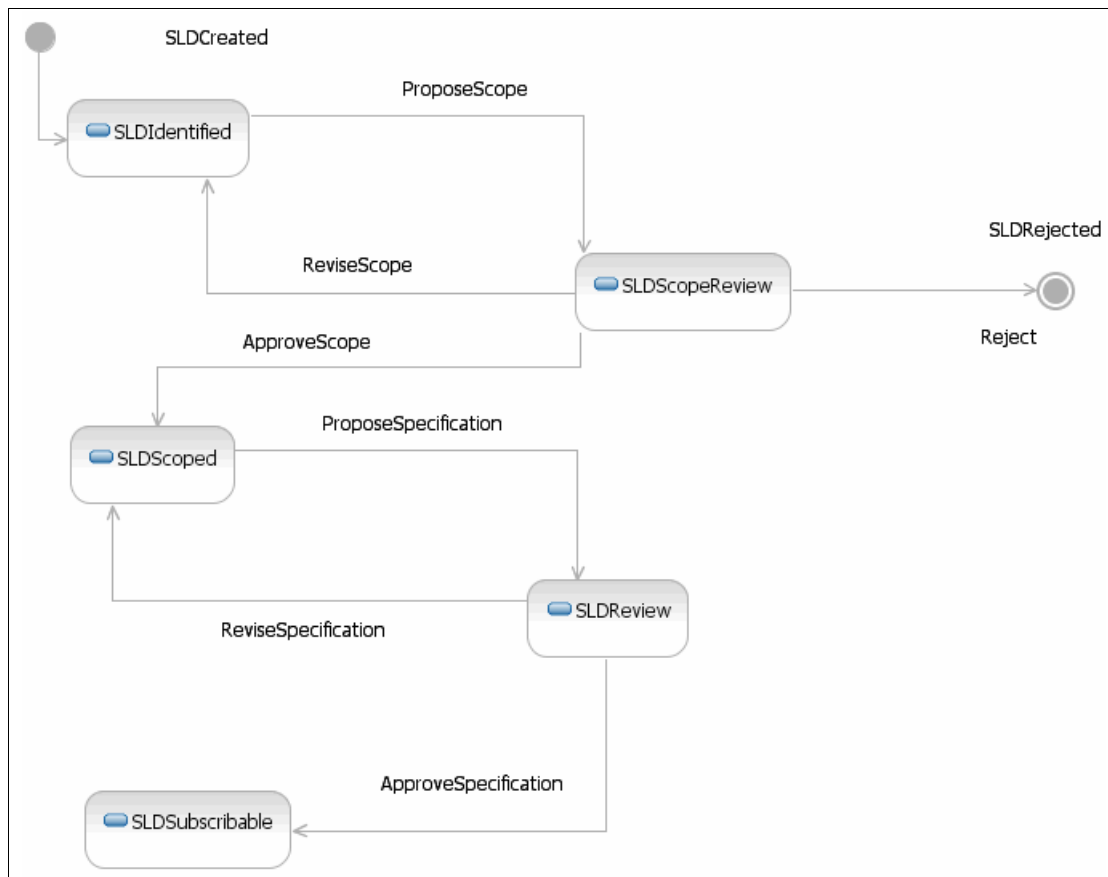


Figure 6-20 SLD life cycle, model, and assemble phases

Table 6-9 describes the states of the model and assemble phases of the SLD life cycle and, for each state, names the transition that moves an SLD entity forward to that state.

Transition note: Where there is a transition that moves an SLD entity from one state back to a previous state, that transition is not listed in Table 6-9. Refer to Figure 6-20 to see all transitions.

Table 6-9 SLD life cycle model and assemble phases explanation

Transition	State	Description
Initial state	SLD identified	A new QoS has been identified that can be configured to work with an existing capability version. The scope and qualities of service that will be made available are defined, and then put forward for scope review.
Propose scope	SLD scope review	In this state, the decision is made to proceed with allocating the resources to develop a new SLD.
Approve scope	SLD scoped	The qualities of service are elaborated and a complete specification defined for the SLD. This specification will allow consumers to develop to the SLD if they want to subscribe to a service and set up an appropriate SLA. After this specification is complete, the SLD goes forward for review.
Propose specification	SLD review	The stakeholders review and approve the SLD, allowing consumers to subscribe and develop against it. For SLDs that are developed as part of the version during its SOA life cycle, this review is coincident with the specification review.
Approve specification	SLD subscribable	SLAs can be established, through the agreed endpoints relationship, to reference this SLD. Development can continue against a subscribable SLD but no interactions can be undertaken with its endpoints yet. Thus, SLAs can be approved and enter the inactive state if the SLD is subscribable but cannot be made active until endpoints become online.

Deploy and manage phases

The *deploy and manage* phases of the SLD life cycle in the governance enablement profile is used to deprecate or supersede an SLD entity and to retire a deprecated SLD when it is no longer in use. Figure 6-21 depicts the transitions and states of the deploy and manage phases of an SLD entity as defined in the SLD life cycle.

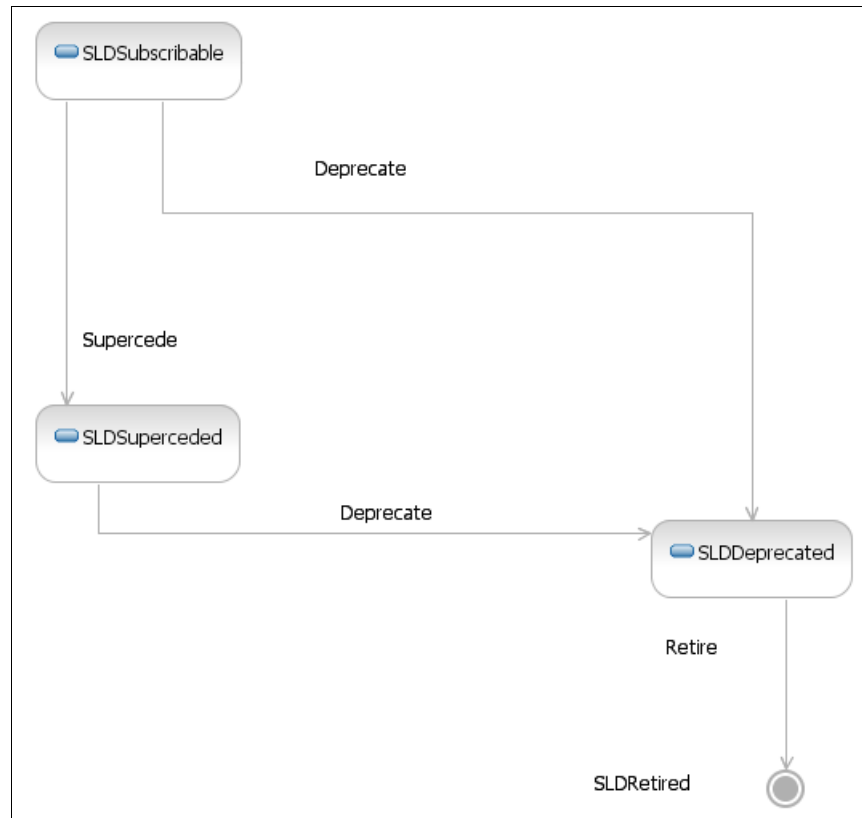


Figure 6-21 SLD life cycle deploy and manage phases diagram

Table 6-10 describes the states of deploy and manage phases of the SLD life cycle and for each state names the transition that moves an SLD forward to that state.

Table 6-10 SLD life cycle deploy and manage phases explanation

Transition	State	Description
Supersede	SLD superseded	A new compatible SLD becomes subscribable, with active endpoints, and the provider wants to move consumers and their SLAs onto this new provided SLD. No new subscriptions can be made to an SLD in this state.
Deprecate	SLD deprecated	All existing SLAs are moved onto the compatible SLDs, and these endpoints are made inactive. Existing SLAs must be renegotiated to directly reference the compatible SLD.
Retire	SLD retired	The SLD has no consuming SLAs.

6.4.5 The SLA life cycle

The SLA life cycle in the governance enablement profile is used to govern an SLA entity from being initially identified, through to being activated, and, eventually, terminated when it is no longer required. Figure 6-22 depicts the transitions and states of an SLA entity as defined in the SLA life cycle.

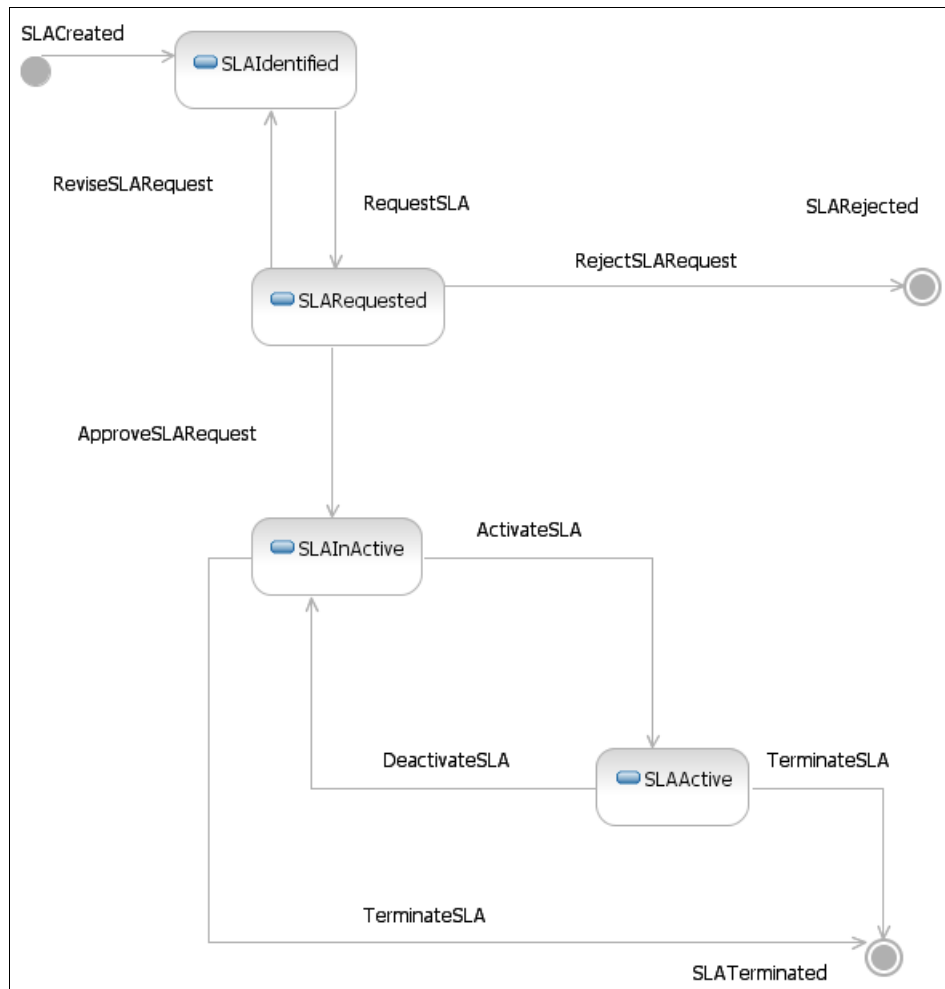


Figure 6-22 SLA life cycle

Table 6-11 describes the states of the SLA life cycle and, for each state, names the transition that moves an SLA entity forward to that state.

Transition note: Where there is a transition that moves an SLA entity from one state back to a previous state, that transition is not listed in Table 6-11. Refer to Figure 6-22 to see all transitions.

Table 6-11 SLA life cycle explanation

Transaction	State	Description
Initial State	SLA identified	This state is entered as soon as a consumer, represented by a capability version, requests a dependency on a service version or other capability version that offers the SLD that they require.
Request SLA	SLA requested	The agreed endpoints relationship target has been selected together with details of the required SLA properties and policies. The provider of the selected SLD must approve the request, reject it, or ask for it to be revised.
Approve SLA request	SLA inactive	The development team that want to consume the service can continue their development based on the consumption of this specific SLA, but they do not yet have authorization to access any endpoints.
Activate SLA	SLA active	All of the approved endpoints associated with the SLD, that are online, can be invoked using the terms of the SLA. There might be situations where the SLA is deactivated, in which case the SLA enters the SLA inactive state and any further interactions are blocked until it is reactivated.
Terminate SLA	SLA terminated	No interactions from this SLA are permitted.

6.4.6 The endpoint life cycle

The endpoint life cycle in this governance enablement profile is used to govern an endpoint entity from being approved for use, through to being retired. Figure 6-22 depicts the transitions and states of a endpoint entity as defined in the endpoint life cycle.

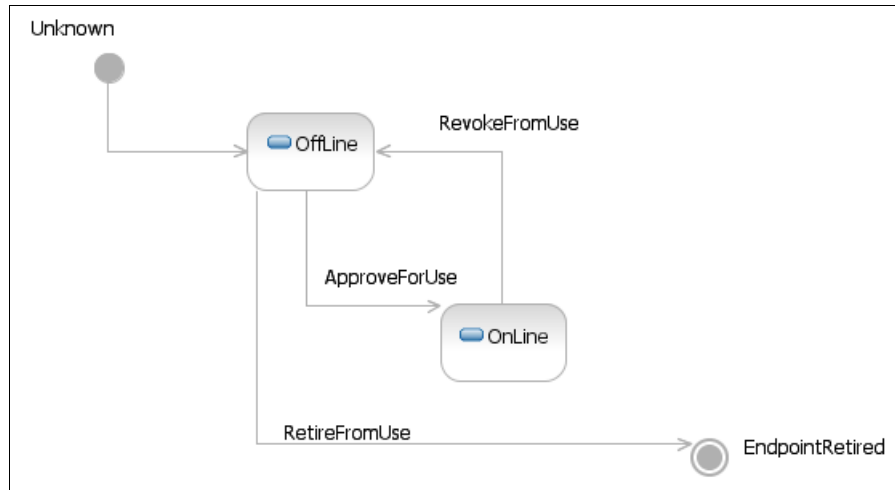


Figure 6-23 Endpoint life cycle

Table 6-12 describes the endpoint life cycle and, for each state, names the transition that moves an endpoint entity forward to that state.

Transition note: Where there is a transition that moves an Endpoint entity from one state back to a previous state, that transition is not listed in Table 6-12. Refer to Figure 6-23 to see all transitions.

Table 6-12 Endpoint life cycle explanation

Transition	State	Description
Initial state	Offline	An offline endpoint might be deployed and thus be reachable by potential consumers, but if protected by mediations that can access the endpoint state, access to this particular endpoint is denied.
Approve for use	Online	Mediations can consider this endpoint as possible routing target.
Retire from use	Endpoint retired	The endpoint has been removed from the environment.

As the entities pass through these various life cycles, constraints as to which role can perform transitions and if these entities must be related to another entity are enforced through *policies*, which we discuss in the next section.

6.5 Policies in the governance enablement profile

The policies in the governance enablement profile specifies which roles (who) can perform a specification action (what) on entities with particular attributes or in a particular state (when). In this section, we give a brief overview of the following policies:

- ▶ Life cycle transition policies

The life cycle transition policies in the governance enablement profile control which user roles can perform transitions on objects that are in a specific life cycle. For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_policies_life_cycle_transition.html

- ▶ Life cycle detail policies

The life cycle detail policies in the governance enablement profile enforce constraints on objects for certain life cycle transitions, and for updates. For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_policies_life_cycle_detail.html

- ▶ Life cycle update policies

The life cycle update policies in the governance enablement profile determine which roles can update or delete objects that are in a specific state in a life cycle. For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_policies_life_cycle_update.html

- ▶ Governance detail policies

Governance detail policies are applied to all entities that are undergoing governance operations. For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_policies_detail_governance.html

6.6 Governance profile taxonomy

WSRR provides the governance profile taxonomy, which represents environments and business domains in an SOA environment. It is included in both the basic profile and governance enablement profile by default. There are two group of classifications included with the profile:

- Environment
- Business domain

The environment classifications are typically assigned to artifacts, such as service endpoints or WSDL documents, to filter promotion of entities to specific runtime environments. The environment classifications can be applied based on the deployment environment states that are defined in the SOA life cycle.

Figure 6-24 shows the UML diagram of the environment classification classes.

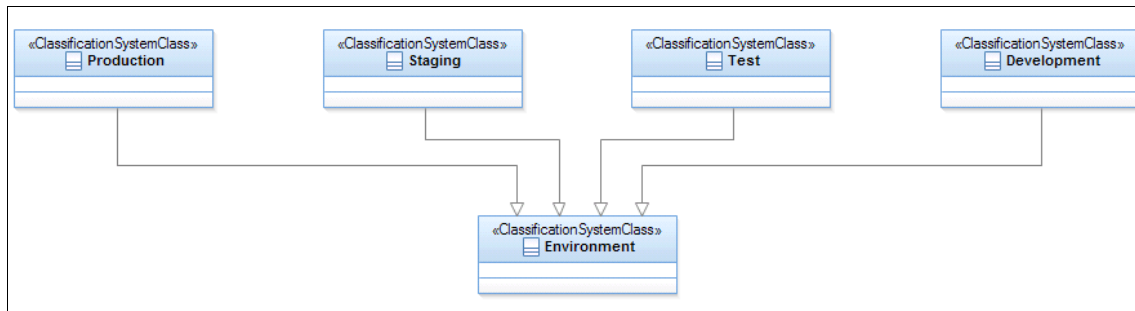


Figure 6-24 Environment classifications of the governance profile taxonomy

Business domain taxonomies are included as examples only. Typically, you define business domain classifications to fit your enterprise. You can use business domain taxonomies to classify service artifacts and metadata appropriately.

Life cycle states of all life cycles in the governance enablement profile are also available as classifications. The life cycle state classification for the current state of an artifact is assigned automatically to the artifact as it is governed and transitioned through its life cycle. Classifying helps you with design time discovery and reuse of services. It also helps with the runtime discovery and endpoint lookup based on classifications and life cycle states.

You typically need to add your own classification systems or modify the provided taxonomy to fit specific aspects of your service environment. You need to develop these taxonomies with input from various business stakeholders and SOA governance, development, and operations roles. Also note that the state of an

artifact in its governance life cycle is assigned automatically with a life cycle state classification.

6.7 Plug-in configurations

WSRR provides plug-ins that use its modifier and notifier APIs to customize behavior and to enforce governance policies and validations. These plug-ins are provided with both the basic profile and the governance enablement profile, and are configured using the WSRR configuration perspective.

The profile includes the following key plug-ins:

- ▶ Governance policy validator
You use the governance policy validator to control the operations that can be performed on specific entities in WSRR, based on the metadata (properties, relationships and classifications) that is attached to those entities.
- ▶ Correlator modifier
You use the correlator modifier plug-in to create WSRR service model objects called *correlator objects*, which correlate and create service model objects and group together various objects in WSRR.
- ▶ Configurable modifier
The configurable modifier is a plug-in that you can use to have WSRR create, modify, or delete objects automatically in response to a user operation.
- ▶ WebSphere ESB policy support
The plug-in provides support for WebSphere ESB policy enforcement in an existing profile that is imported from WSRR Version 6.2 or earlier.
- ▶ WS-I validator
The WS-I validator checks that WSDL documents stored and managed in WSRR conform to the Web Services Interoperability (WS-I) compliance rules.

Note: You can write your own code that WSRR calls automatically during standard processing. You implement code in Java, referred to as a WSRR custom plug-in, using WSRR APIs in Java.

In this section we concentrate on the correlator modifier.

6.7.1 The correlator modifier

The correlator modifier correlates and creates appropriate service model entities and relationships as multiple WSDL documents are loaded. This modifier facilitates keeping track of the same logical service version. In this section, we illustrate which WSDL logical entities are created when a WSDL is loaded and show how the service model entities are created and related to the WSDL logical entities.

For details about the correlator modifier, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_correlator_modifier_R5.html

Let us assume a service is defined in multiple WSDL documents. The WSDLPortType element is defined in an interface WSDL file. The binding and service elements of the service are defined in a second WSDL document that imports the interface WSDL document.

In Figure 6-25, the WSDL document that contains only the WSDLPortType element (no service or binding elements) is loaded into WSRR. Only the service interface entity of the service model is created in WSRR.

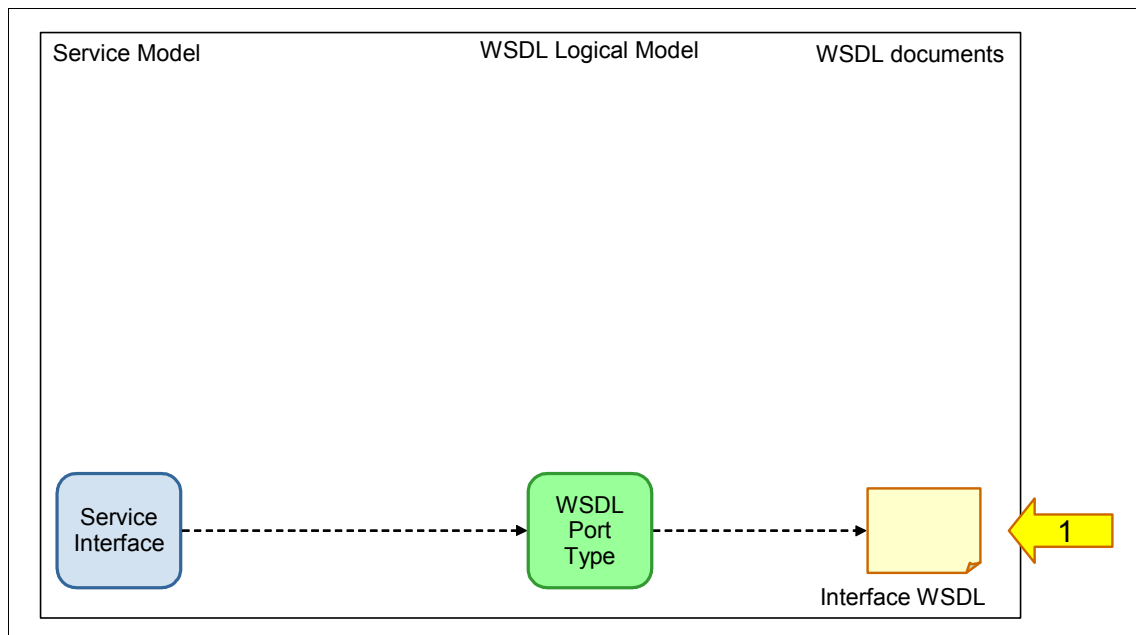


Figure 6-25 WSDL that contains only an interface element is loaded

Next, a WSDL document that contains the binding and service elements of a WSDL definition is loaded, as shown in Figure 6-26. This document imports the interface WSDL that was loaded in the previous step. The other elements of the service model are created appropriately and are related to the corresponding logical entities by the correlator modifier.

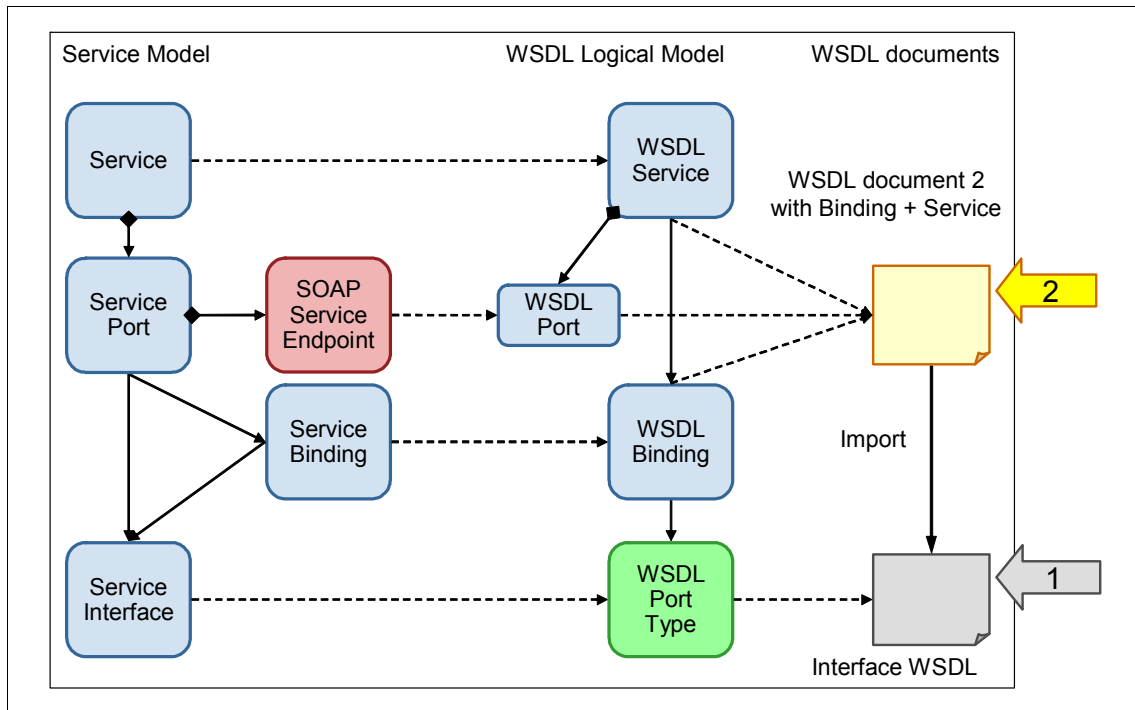


Figure 6-26 A WSDL document containing the service and binding elements of a WSDL definition is loaded

In Figure 6-27, the third and fourth WSDL documents are loaded. Each document contains the service and binding elements and imports the interface WSDL that was loaded during the first step. The name, namespace, and version of the service and binding elements are the same as the prior service and binding WSDL document.

The correlator modifier correlates the logical WSDL service and WSDL binding entities to the existing service and service binding entities of the service model. However, the name of the port element is different. Each port contains a different location address for three different deployment environments. Thus, three service endpoint entities corresponding to three WSDL port logical elements are created by the correlator modifier with the appropriate relationships between them.

Our discussions here use correlation in loading WSDL documents as an example. The correlator modifier performs similar actions when XSD documents or an SCA modules are loaded.

For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_correlator_modifier_R5.html

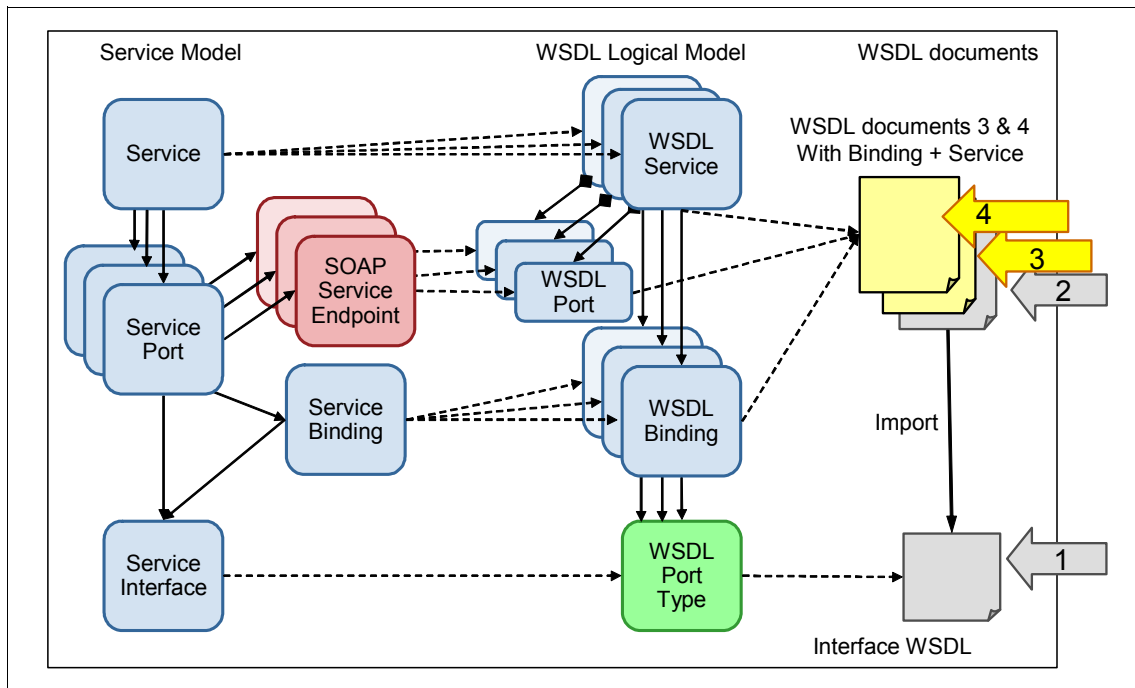


Figure 6-27 Service model entities created and correlated by the correlator modifier are loaded

When a WSDL document is deleted, all the related WSDL logical entities are deleted automatically. However, corresponding service model entities remain without the WSDL document that created it. If you later reload the WSDL that you deleted (perhaps with modifications of its content that do not change the correlated properties), the service model is related to appropriate WSDL logical entities that are created as the WSDL is loaded.

Thus, the correlator modifier keeps the conceptual service model entities and relationships consistent as you load different elements and versions of WSDL documents. In particular, it correlates WSDL logical service elements to the *service* entity in the service model to maintain a consistent version of a service.

Creating service model elements manually: You can create service model elements manually without loading WSDL. For example, you can enforce naming standards on interface WSDLs by configuring governance policies that allow:

- ▶ Creating a service interface by persons in an SOA governance role only
- ▶ Loading a WSDL by persons in other roles only if it correlates to the existing service interface

This method allows only WSDLs with properties that adhere to the naming standards, such as name and namespace.

6.8 Web user interface configuration

WSRR comes with a set of XML configuration files to define views and perspectives for different roles of a web user console and Business Space user interface.

6.8.1 WSRR web user console

The default WSRR web user console designed for the governance enablement profile is defined in a set of XML documents. There are seven different types of XML definition files that are used by the WSRR web user interface. The user interface is organized into *perspectives*, which are a collection of views that display the data in WSRR that is catered to a user role.

A perspective associates a menu bar definition, a navigation tree definition, a set of detail view definitions, and a set of collection view definitions. Each perspective has its own home page, which provides a range of functions to make it easier to find, and work with, WSRR objects.

For details about configuration files for the web user console, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.webui.doc/html/config_admin_webui_configuration.html

The governance enablement profile provides perspectives for configuration, administrator, general users, and its four governance enablement profile user roles:

- ▶ Business
- ▶ SOA governance
- ▶ Development
- ▶ Operations

6.8.2 WSRR Business Space user interface

A Business Space for different user roles can be created using configuration templates that group different WSRR Business Space widgets in related pages and display views. For a list of Business Space widgets for WSRR, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_business_space_widgets.html

By default, the governance enablement profile comes with Business Space configurations for the business role (Service Registry for Business) and the SOA governance role (Service Registry for SOA Governance). You can customize your Business Space for other user roles and save them as configurations that you can share.

You can create your own Business Spaces using these configurations as templates. You can then customize these WSRR pages and widgets for a user role and save them as configuration templates to be included in the WSRR configuration profile. For details, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_business_space_and_wsrr.html

6.9 Modifying the governance enablement profile

In various cases, you might need to further customize the governance enablement profile to meet the specific requirements of your SOA environment. The following customizations are typical:

- ▶ Customize access control roles and add or remove related permissions
- ▶ Add classification systems that represent your enterprise environment, such as business domains, architecture, application domains, and operational environments

- ▶ Customize web user interfaces to add perspectives for the new roles you might have added, add task menus, modify view queries, and so forth
- ▶ Remove or add a state to a life cycle to meet your governance processes.
- ▶ Add new WSRR governance policies to those provided in governance enablement profile or remove various existing policies to meet your own control requirements

WSRR Studio provides convenient facilities to perform key customizations to the governance enablement profile. You can use UML class diagrams and inheritance models to create classifications. You can modify a life cycle by modifying the corresponding state diagram and transitions in the studio. You can author transition policies and specify them at a particular state of a life cycle. You can then synchronize the customized version of the governance enablement profile in the studio to your WSRR installation.

For details about using WSRR Studio, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_new_75_studio.html



Registering services

This chapter introduces user interfaces for working with WebSphere Service Registry and Repository (WSRR) and focuses on the Business Space user interface for WSRR. The chapter starts with an overview of the Business Space user interface and the WSRR widgets that are available to support the management of your governance environment.

This chapter also provides step-by-step instructions for registering an existing service used by our fictional supply company into WSRR using the governance enablement profile.

7.1 WebSphere Service Registry and Repository user interfaces

WSRR provide two user interfaces. One interface is the traditional web UI that is configured as a standard part of WSRR installation. You can use the WSRR web UI to use and manage all aspects of WSRR from a compatible web browser.

WSRR web UI: For more information about using the WSRR web UI to manage your WSRR environment, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.webui.doc/html/home_page.htm

The other user interface provided for WSRR is Business Space. Business Space is a browser-based GUI that lets business users interact with content from products in the WebSphere portfolio. The spaces that you create are collections of related content that can provide you with insight into your business and the capability to react to changes in it.

Business Space now supports a number of WSRR specific widgets and two role-based space templates:

- ▶ Service Registry for Business
- ▶ Service Registry for SOA Governance, based on the governance enablement profile model

A third space, Service Registry for Policy Analytics, is also available for viewing data related to the enforcement of governance validation policies.

7.2 Using the Business Space interface with WSRR

Business Space is a rich Web 2.0 user interface framework based on MashUp technology.

In Business Space, a *space* consists of pages that you define or create based on preconfigured templates provided by each WebSphere product. (WSRR provides three templates.) Each *page* can contain one or more *widgets* that are configured to perform business tasks. Many WebSphere products provide preconfigured widgets, each with specific functions.

You can have many spaces, with each one having a separate purpose. For example, you can use a space with widgets from WSRR to create and view items and data in the service registry.

After installing Business Space, you need to create the WSRR spaces. To open the Business Space user interface, enter one or other of the following URLs in your web browser, depending on whether security is enabled on the WSRR server that is configured as the endpoint for the WSRR widgets, where hostname and port are the host name and port value of the WSRR server.

- ▶ Security enabled:
`https://hostname:port/BusinessSpace`
- ▶ Security not enabled:
`http://hostname:port/BusinessSpace`

Figure 7-1 shows the Business Space UI welcome page layout. From the welcome page you can access tutorials and information about how to get started with constructing your spaces, or you can go directly to spaces that you might have already created.

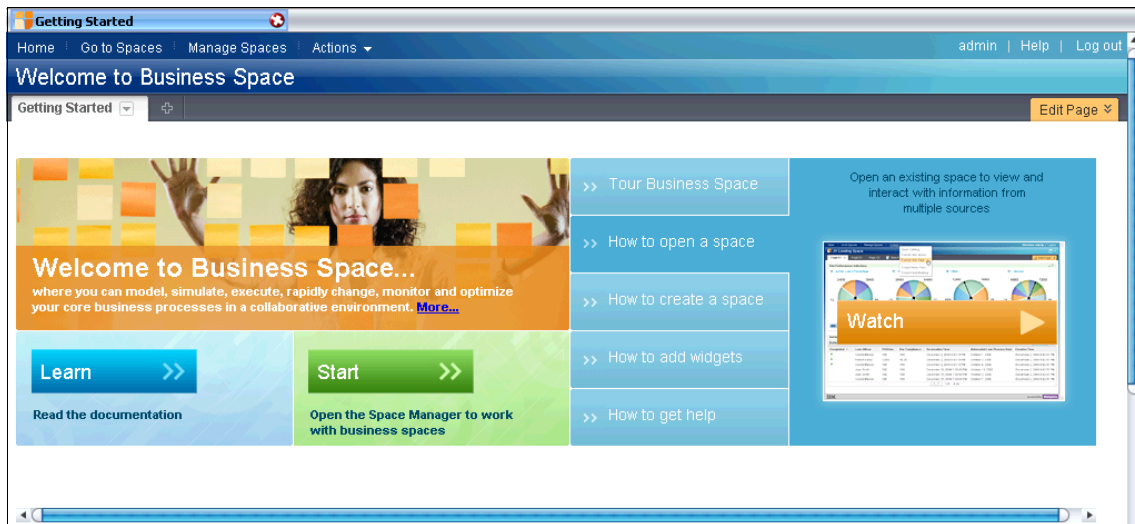


Figure 7-1 Business Space Welcome page

7.2.1 Service Registry Business Space widgets

Widgets in Business Space are the pluggable user interface components that you use to define the functionality of your business spaces.

Various widgets communicate with other widgets so that events in one widget affect the contents of another widget. You can minimize, maximize, and drag widgets while laying out a Business Space page. In addition, each widget has a menu that contains actions available for that widget. Business Space provides a widget palette that contains categories of widgets that you can use to configure the pages in spaces. The following WSRR categories are available in the widget palette:

- ▶ Service Registry
- ▶ Service Registry Policy Analytics

Business Space provides 14 widgets for various Service Registry functions. You can add or remove widgets to pages based on user role or preference.

To add or remove widgets from pages, click **Edit page**, towards the upper-right corner of the Business Space UI window. The widget palette opens (as shown in Figure 7-2). You can search for widgets by name or use the drop-down menu option to search by type. Widgets are classified under certain groups, for example Service Registry Widgets and Policy Analytics Widgets.

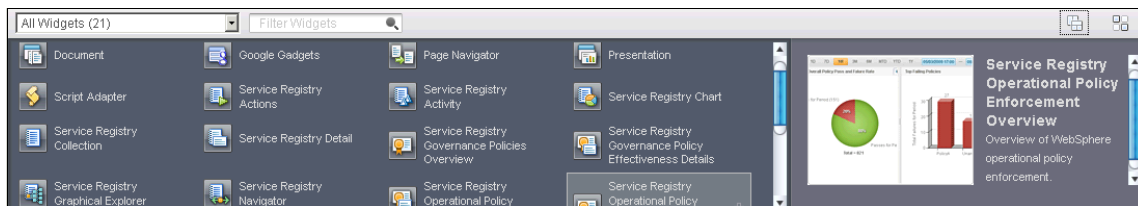


Figure 7-2 Business Space Service Registry widgets

Service Registry widgets: For a complete list and description of all the available Service Registry widgets for Business Space, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/twsr_managing_policy.html

The following Business Space widgets simplify the publication and governance of services and increase visibility:

- ▶ Service Registry Navigator widget

This widget provides the ability to browse registry content visually, giving a one-upstream and one-downstream view of related content.

- ▶ Collection View widget

This widget allows the creation of new views that you configure with a wizard. You specify a single query or an aggregation of multiple queries to create

other Watch Lists, which can appear on your space as separate widgets or in the view menu of a single Collection View widget. The Collection View widget has an icon mode that enables you to browse the collection result by icons, thus enabling quicker recognition of the content type for which you are looking.

- ▶ Service Registry Actions widget

This widget enables you to configure your own actions, allowing your governance process to be more prescriptive for users.

- ▶ Details widget

This widget enables you to configure use visibility of properties for each registry type, and to group those properties into custom sections. You can determine which classification taxonomies are displayed to the user. You can also determine whether a property is visible on creation. The visibility of relationships can also be configured, and custom queries can be included as relationships, thereby allowing you to have focused content for each user role.

- ▶ Service Registry Graphical Explorer widget

This widget provides impact analysis of the registry content in visual form with icons that represent the various types of content that exist in the registry repository. It allows you to quickly understand all the relationships.

- ▶ Activity widget

This widget offers an efficient way to keep track of changes that are made and those who are responsible for making the changes.

7.2.2 Customizing and configuring the Business Space UI and Service Registry widgets

You can create customized Business Space configurations for use by users with particular roles. You can create new spaces based on the available WSRR templates and then customize the widgets that are used in this space. After the new customized configurations are ready, you can save them as objects in WSRR. The configuration for all WSRR widgets within a business space is stored in a single WSRR Configuration object.

For example, you can use customization and configuration features to set up a user space where the widgets are configured to follow these rules:

- ▶ A search widget offers only searches for specified types of object in WSRR.
- ▶ An actions widget offers only a subset of actions.
- ▶ A details widget displays only selected details of objects.
- ▶ A collection widget displays only selected collections.

Alternatively, you can create preconfigured spaces for certain user roles, such as:

- ▶ Business analyst
- ▶ Operations professional
- ▶ SOA governance professional

Consider the following important points when configuring and customizing spaces with WSRR:

- ▶ Each space you create points to a single WSRR space configuration, which is saved as an object in WSRR.
- ▶ Each WSRR widget within the space can be configured in Business Space using its Edit Settings window.
- ▶ Widget settings are saved to the space's WSRR configuration. Thus, widgets of the same type (for example, two search widgets) share the same configuration.
- ▶ Various widgets have settings that are not saved to the WSRR configurations and act on a per-instance basis. In these situations, you are told that the settings are on a per-instance basis.
- ▶ As a user, inside Business Space, you can switch to another WSRR Space Configuration, which changes the entire space's settings.
- ▶ WSRR provides ready-configured space configurations. Each WSRR template has a default space configuration that is selected when the template is used to create a space. You can change the configuration later.
- ▶ When creating custom spaces, you can copy the predefined WSRR space configurations and then make personal modifications to them.

Configuring the Business Space UI: Configuring the Business Space user interface is described in more detail at the following site:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_display_settings_graphical_explorer_widget.html

7.2.3 Using the Service Registry for SOA governance template

The Service Registry for SOA Governance template is aimed at SOA governance professionals. It creates four pages consisting of widgets that an SOA governance professional can use to search, browse, and view collections and includes details of items in the registry. This template allows you to perform key tasks and to view service reuse data. You can also view data that is related to the enforcement of governance validation policies.

This template provides the following pages:

- ▶ Overview
- ▶ Browse
- ▶ Graph
- ▶ Governance Policies

Overview page

This page gives an overview of the business objects in your service registry. It shows you approved Business Capabilities and a report of current service reuse levels. It enables you to search for particular objects and to create new actions. The Overview page contains the following widgets:

- ▶ Search widget
This widget is configured to allow you to search for the types of item of most interest for SOA governance.
- ▶ Actions widget
From this widget you can create new instances of items that are useful in SOA governance (for example, Business Service, Business Process, Business Application, Service Version, Process Version, Application Version, and Organization).
- ▶ Service reuse widget
This widget shows a chart that illustrates service reuse in the service registry.
- ▶ Collection widget showing watch list
This widget is configured to show the types of service registry items in which a SOA governance user might be interested.
- ▶ Collection widget showing approved business capabilities
This widget is configured to show business service, business application, and business process items that have an approved governance state.

Figure 7-3 shows the Overview page.

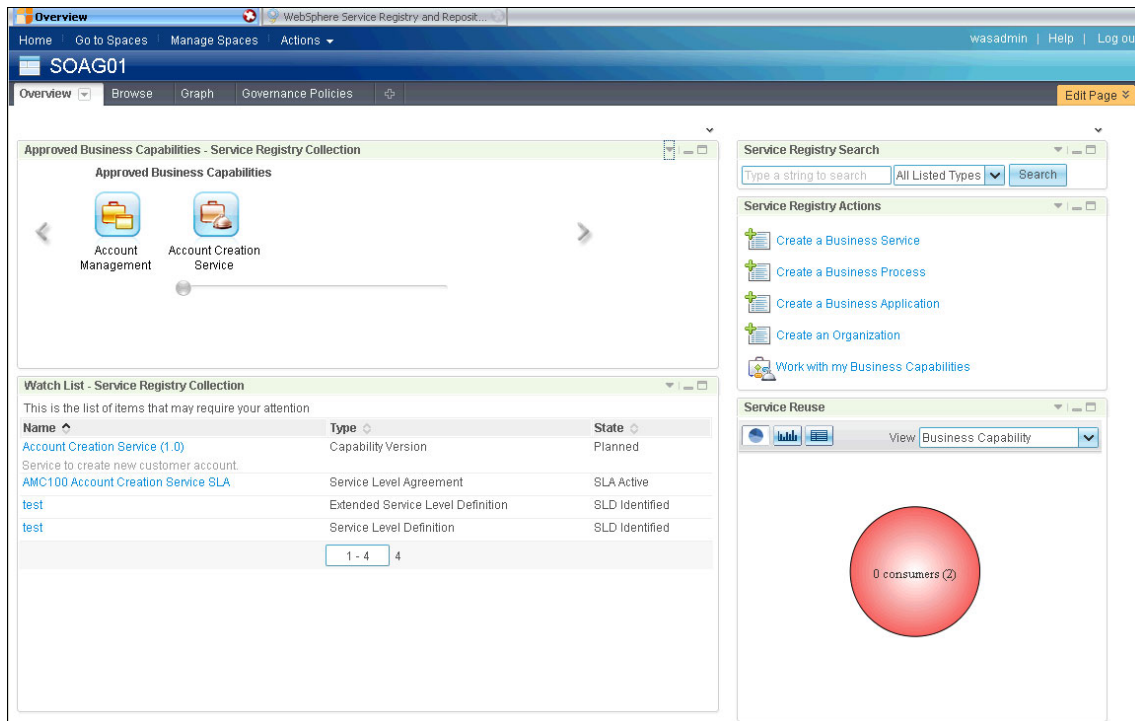


Figure 7-3 SOA Governance template: Overview page

If you select an item in this page, you are switched to the Browse page to view details of that item.

Browse page

This page is used to browse the business objects in your service registry. You can search for objects and view and edit object details. You can also use the navigator to follow the relationships of your object and to select new objects. The Browse page contains the following widgets:

- Search widget

This widget is configured to allow you to search for the types of item of most interest.

- Collections widget

This widget is configured to show recently created items. There are a number of other preconfigured views that you can select.

- ▶ **Navigator widget**
This widget shows a one up, one down graphical view of the item currently displayed in the Detail widget and its relationships.
- ▶ **Detail widget**
This widget shows details of the item currently selected in the Collection widget and enables you to edit the details of that item. You can also choose to view the item and its relationships in the Graphical Explorer widget from here.
- ▶ **Activity widget**
This widget shows recent changes that have been made to items.

Figure 7-4 shows the Browse page.

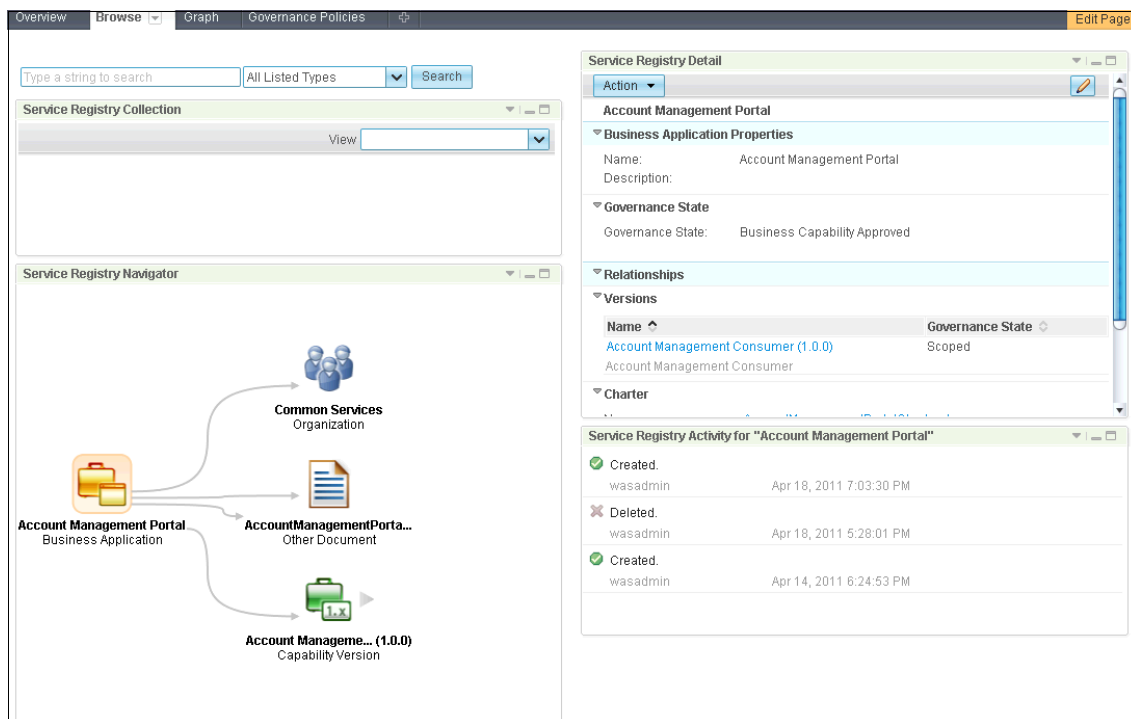


Figure 7-4 SOA Governance Template: Browse page

Graph page

From the Graph page, you can view a graphical representation of business objects and their relationships. You can navigate along the relationships in this view using a Graphical Explorer widget. This widget shows a representation of the item that is selected in the Service Registry Detail widget and the source and target items of its relationships. It provides a graphical representation of how

business objects are interrelated. Preferences specify the depth of the relationships that are displayed. If there are more relationships than can be displayed in the specified depth, an arrow indicates that there are more items to view.

Figure 7-5 shows the Graph page.

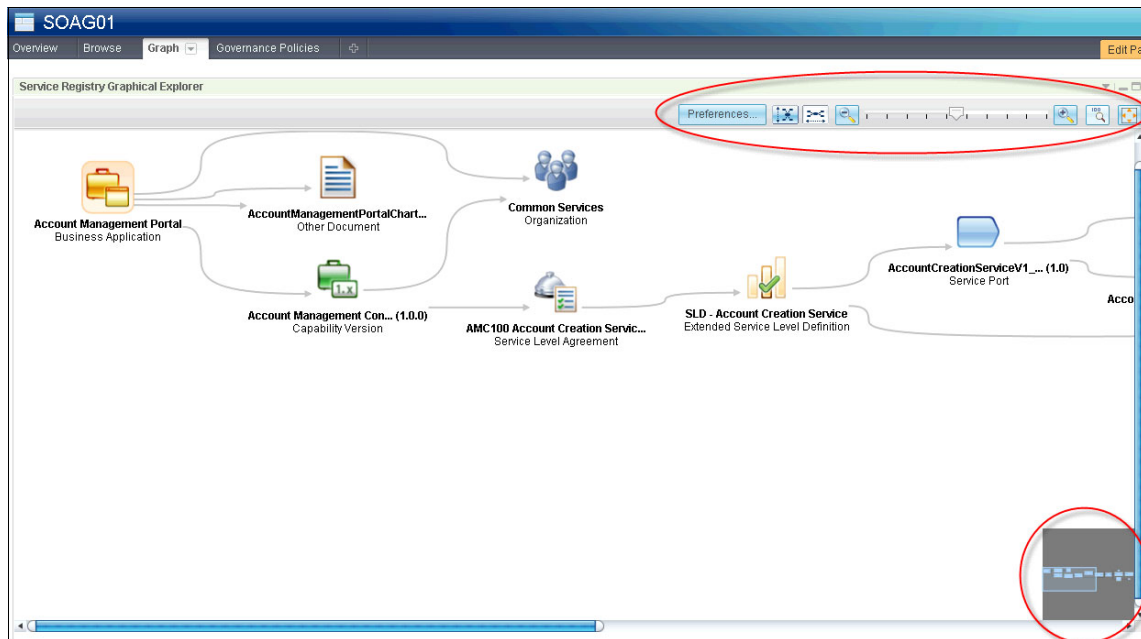


Figure 7-5 SOA Governance Template: Graph page

Graphical Explorer widget appearance: You can specify the appearance of the instance of the Graphical Explorer widget that you are using from the Preferences window. Use this window to set features such as orientation, how items are represented, and the style and color of lines that represent relationships. For more information, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_prefs_graphical_explorer_widget.html

You can specify the appearance and actions of all instances of the Graphical Explorer widget by using the Service Registry Graphical Explorer Display Settings window. For more information, refer to:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/cwsr_display_settings_graphical_explorer_widget.html

If your Business Space configuration was edited to filter out a particular item type, objects of that type are not visible in the Graphical Explorer. In addition, the Graphical Explorer does not indicate that there are more filtered objects other than those items that are currently visible in the Graphical Explorer.

To open a view in the Graphical Explorer:

1. Display the details of the object on which you want to focus in the Detail widget.
2. From the Detail widget, click the **Action** menu, and select **View in Graphical Explorer**. (If you cannot see that item in the menu, open the Detail widget display settings window and enable the Graphical Explorer for the current object type.)

If a relationship has several targets, then the Graphical Explorer widget stacks the targets so that they do not take up too much space. You can specify the number of targets that cause stacking, but the default is five. You can click a target stack to display a list of the targets.

When you point to an icon, a pop-up window displays information about the object, including the object's name, description, and current governance state. The pop-up window also contains a **View Details** link and a **Refocus** link. Click **View Details** to switch to the Detail widget displaying details of the object. (If you are using the Business or SOA Governance templates, this action takes you to the Browse page.) Click **Refocus** to focus on that object to visualize its relationships.

Figure 7-6 shows an example of a Graphical Explorer window.

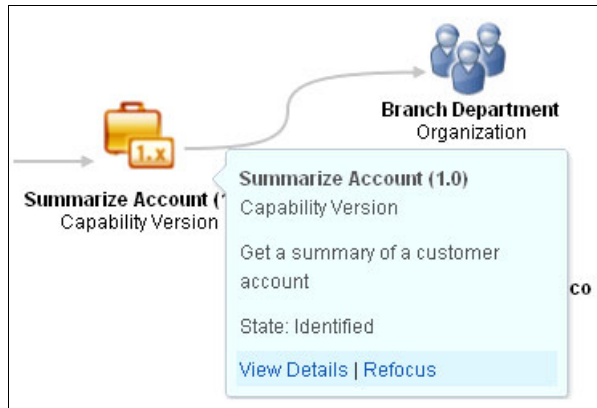


Figure 7-6 Graphical Explorer pop-up window

Clicking an icon switches to the Detail widget and displays the details for that object.

Governance Policies page

Use the Governance Policies page to view data relating to the enforcement of policies that are defined in the governance policy validator. The Governance Policies page contains the following widgets:

- Governance Policies Overview widget
- Governance Policies Effectiveness Details widget

Figure 7-7 shows the Governance Policies page.

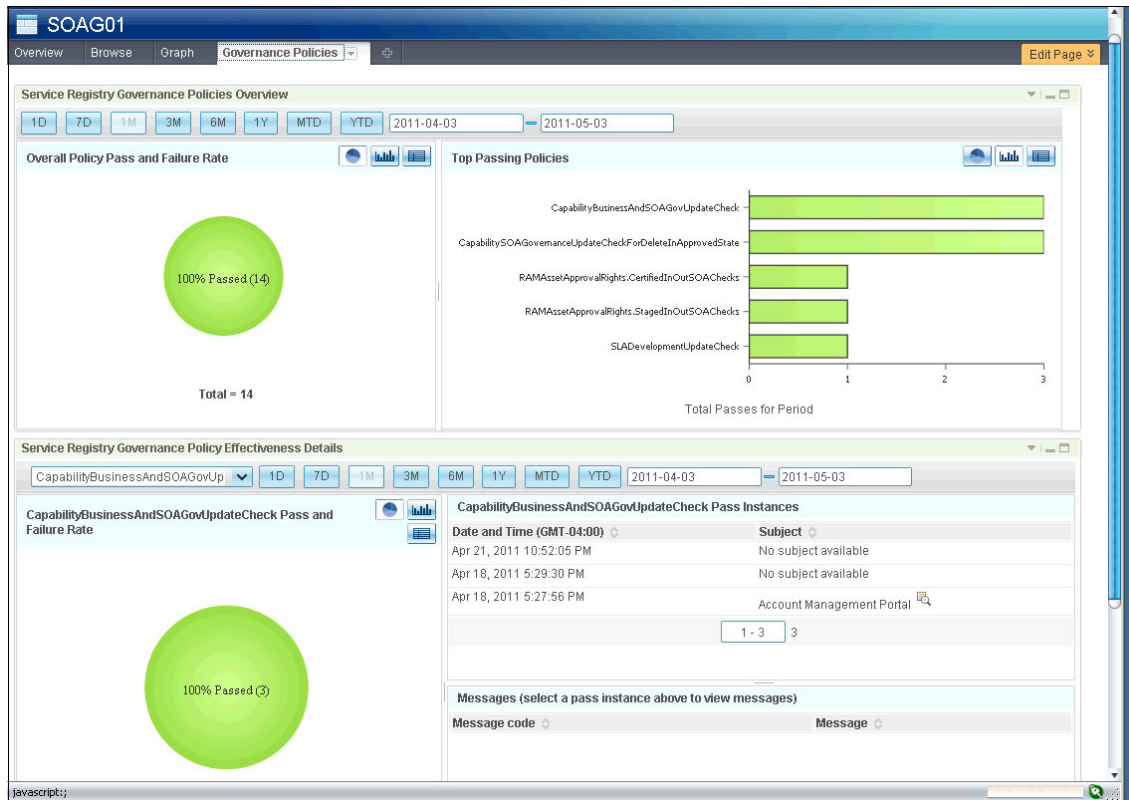


Figure 7-7 SOA Governance Template: Governance Policies page

7.3 Preparing Business Space for use with WSRR

In this section we create and customize a new space using the Service Registry for SOA Governance template.

7.3.1 Creating a new space in Business Space

To create a new space in Business Space based on the Service Registry for SOA Governance template, follow these steps:

1. Log on to the Business Space web UI.
2. At the Welcome Page, click **Start** to access the Space Manager.
3. In the Space Manager window, click **Create Space**.
4. In the Create Space window, specify a Space name of SOAG, select **Create a new space using a template**, and select the **Service Registry for SOA Governance** template, as shown in Figure 7-8.

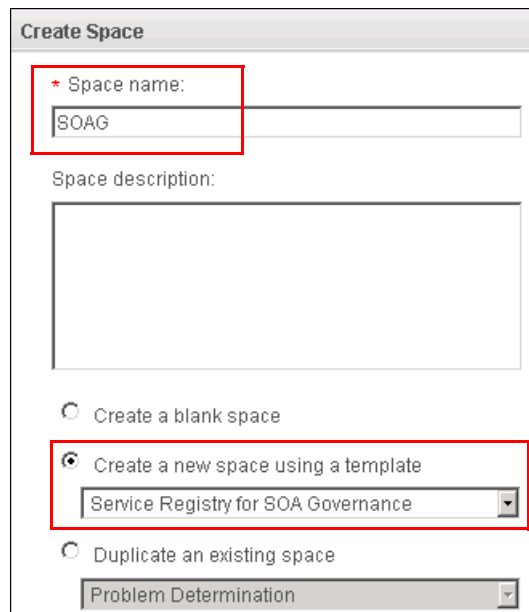


Figure 7-8 Creating a new space

5. Select **Save** to save your new space.
6. In the Space Manager, click **SOAG** to open the new space.

7.3.2 Customizing Business Space to add a Load Documents widget

Services are typically loaded by using the WSRR web user interface. However, we can use the flexibility of Business Space to customize our space, incorporating the Load Documents page of the WSRR user interface into our space. Use the Web Site widget as follows:

1. In the Browse page of the SOAG space, click **Edit Page**; see Figure 7-9.



Figure 7-9 Edit Page button

2. From here, add additional widgets to the page. In the list of widgets, select the **Web Site** widget and click the plus sign (+) to add the widget to the page, as shown in Figure 7-10.

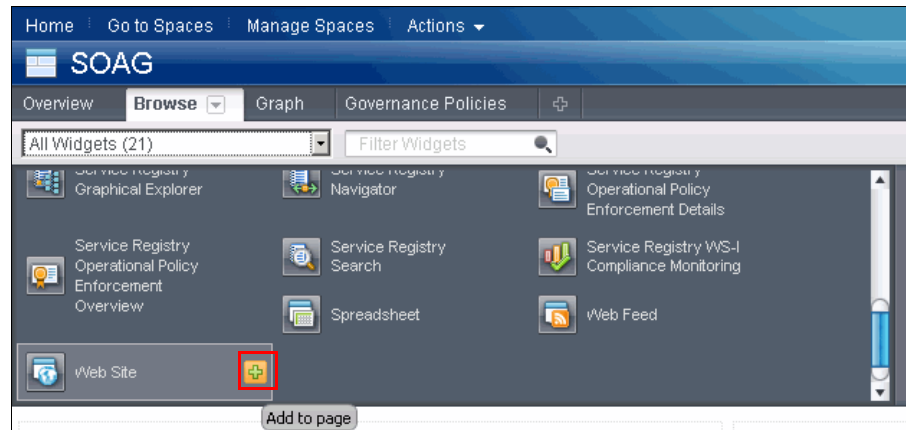


Figure 7-10 Adding the Web Site widget to the page

3. A Web Site widget is added to the page. Select **Edit Settings** to edit the settings for this widget, as shown in Figure 7-11.



Figure 7-11 Editing Settings menu option for the Web Site widget

4. In the Web Site widget, enter the URL that points to the Operations perspective of the WSRR web user interface, as shown in Figure 7-12. We entered the following URL:
`https://localhost:9443/ServiceRegistry/SelectPerspective.do?operation=changePerspective&perspectiveName=GEPOperationsPerspective`
5. Click **Apply**.



Figure 7-12 Entering the URL of the WSRR Operations perspective

6. If you encounter a dialog box that suggests the URL is invalid, click **OK**. The Operations perspective of the WSRR web user interface opens.
7. In the Web Site widget, select **Rename**. Change the Widget name to Load Documents, as shown in Figure 7-13, and click **Save**.

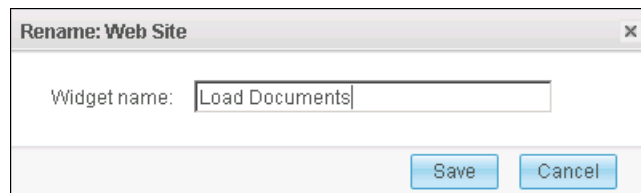


Figure 7-13 Changing the name of the widget

8. Near the top of the page, click **Save**, and then click **Finish Editing**. A new widget called Load Documents now displays in the Details page of the SOAG business space, as shown in Figure 7-14. You might be required to log in.



Figure 7-14 Completed Load Documents widget

7.4 Registering a service using Business Space

This section provides step-by-step instructions explaining how to register a service into WSRR using Business Space. Here, we execute the service registration process based on the governance enablement profile model described in Chapter 6, “Governance enablement profile” on page 143.

This section also discusses how to load service documents, including WSDLs and schemas, and how to annotate each document with metadata, properties, and relationships using the Registry Widgets provided in Business Space.

In this section, we register services for our supply company case study and use these services in later chapters throughout the book. This section contains the following topics:

- ▶ Creating an organizational structure
- ▶ Identifying the business capability provided by a service
- ▶ Registering a service
- ▶ Providing scope and planning information for a service
- ▶ Providing a specification and service level definition for a service
- ▶ Governing the service consumer
- ▶ Completing the services life cycle

7.4.1 Creating an organizational structure

When multiple composite applications use a service, it is vital for effective governance to determine who is responsible for that service. Often an enterprise organizes its staff reporting structure and finances around business operations. The department that is responsible for certain business operations can also be responsible for the development of services and the runtime IT for these operations.

In a service-oriented architecture (SOA), however, another organization can be responsible for the development of the services and runtime IT for given operations. Therefore, services and composite applications in an SOA often do not follow an enterprise's strict hierarchical reporting and financial structure, thereby creating gaps and overlap in IT responsibilities.

In our case study, the supply company recently implemented WSRR for the governance of SOA services. In addition, it opted to take advantage of the predefined structure that is provided in the WSRR governance enablement profile.

As such, part of the administration setup calls for an administrator to create organizations that are represented in the registry. These are required to allow relationships to be built between services and organizations, to show service ownership and consumption.

Before we populate the WSRR with WSDL documents, we create the following organizations:

- ▶ A supply company, which is a top-level organization that represents the complete enterprise
- ▶ A common services organization, which is a child organization of the supply company that represents the departmental team responsible for the development and delivery of services that are shared throughout the enterprise
- ▶ The commercial group, which represents the commercial line of business (LOB).

Creating an organization

To create the top-level organization:

1. Ensure you have the SOAG space in Business Space open.
2. On the Overview page of the SOAG space, note the Service Registry Actions widget. From here you can create services, processes, applications, and organizations as shown in Figure 7-15.



Figure 7-15 Service Registry Actions widget

3. In the Actions widget, click **Create an Organization**.
4. In the Create an Organization window enter the name: Supply Company, as shown in Figure 7-16.

 A screenshot of a "Create an Organization" dialog box. It has a title bar with a close button. The main content area contains instructions: "Create a new entity of type: Organization. When you have specified all required property values, and relationship targets, click 'Finish'." Below this is a section titled "Organization Properties" with a dropdown arrow. Under this section are four labeled input fields: "Name:" (highlighted with a red rectangle and containing "Supply Company"), "Description:" (a larger text area), "Contact:", and "Contact Email:".

Figure 7-16 Creating the Supply Company organization

5. Click **Finish**. The Browse page opens. The Service Registry Detail widget displays the new organization, as shown in Figure 7-17.

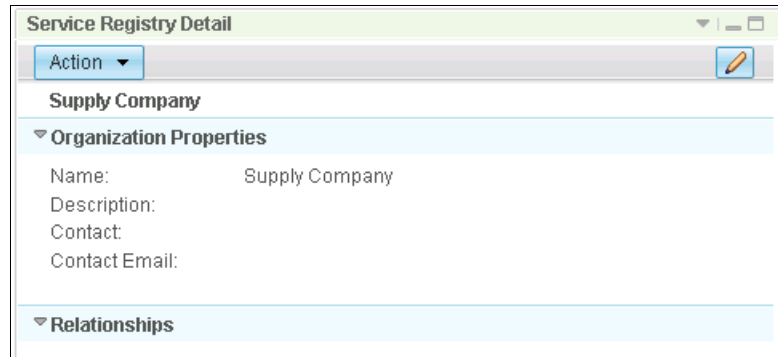


Figure 7-17 New organization added

Creating child organizations

To create the child organizations for the supply company:

1. In the Service Registry Detail widget, click the Edit icon (Figure 7-18) to edit the metadata for the supply company organization.



Figure 7-18 Edit icon

2. In the Edit: Supply Company window, under Child Organizations, click **Add Organization**.
3. Specify the name: Common Services and then click **Create**. In the Create: Organization window, click **Finish**.
4. Repeat the previous step to create a second child organization of the supply company called Commercial. Two child organizations of Supply Company are now defined, as shown in Figure 7-19.

Edit: Supply Company

Organization Properties

Name:

Supply Company

Description:

Contact:

Contact Email:

Relationships

Child Organizations

Name ^	Contact	Contact Email
Commercial		
Common Services		

+ Add Organization

Figure 7-19 Child organizations added to supply company

- Click **Finish**. The Service Registry Navigator displays the three organizations, as shown in Figure 7-20. You might have to refresh the page.

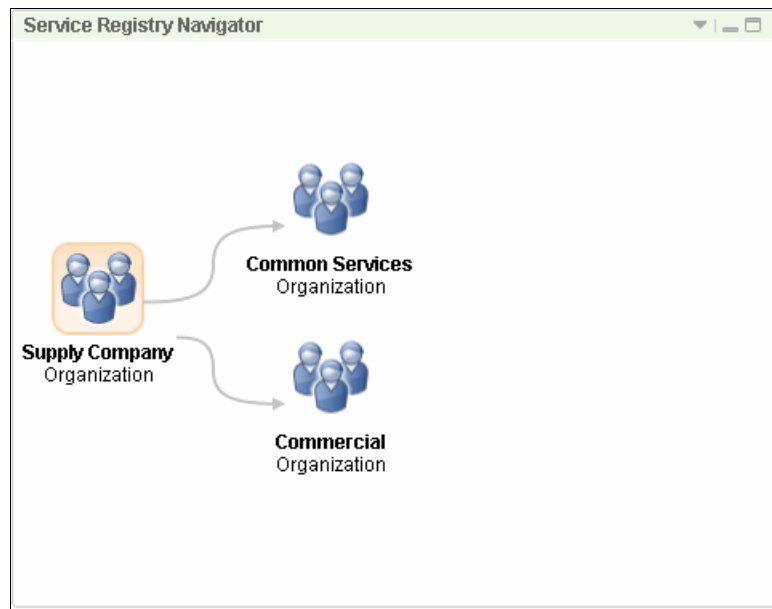


Figure 7-20 Navigator showing relationships between the three organizations

7.4.2 Identifying the business capability provided by a service

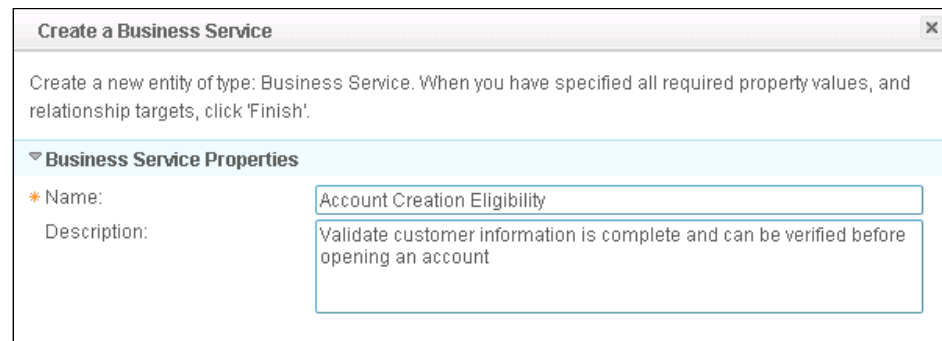
An important part of governing a service is to identify how it adds value to the business. WSRR uses business capabilities to define the business view of a service.

If there is no existing business capability that the service realizes, you must create and define a business service. You need to add the following key defining components:

- ▶ A service charter to define the scope and role of the business service and an owning organization
- ▶ The owning organization that is responsible for defining requirements for the capability and owning any realizations of this capability, such as the Account Creation service that has now been loaded

To identify the business capability provided by a service:

1. Create a business service.
 - a. On the Overview page of the SOAG space, locate the Service Registry Actions widget. In this widget, click **Create a Business Service**.
 - b. In the Create a Business Service window enter the following information, as shown in Figure 7-21:
 - Name: Account Creation Eligibility
 - Description: Validate customer information is complete and can be verified before opening an account



Create a Business Service

Create a new entity of type: Business Service. When you have specified all required property values, and relationship targets, click 'Finish'.

▼ Business Service Properties

* Name: Account Creation Eligibility

Description: Validate customer information is complete and can be verified before opening an account

Figure 7-21 Creating a business service

2. Define the scope of the Account Creation Eligibility business service by loading a charter:

- a. Still in the Create a Business service window, under the Charter relationship, click **Add Other Document**.
- b. Click **Browse**, and select **AccountCreationServiceCharter.doc**. Click **Open**.

AccountCreationServiceCharter.doc: You can find this document in the additional material that is supplied with this book in the \Register_Service directory.

- c. Click **Load**. The charter document is added as a target of the Charter relationship, as shown in Figure 7-22.

The screenshot shows a window titled "Create a Business Service" with a close button (X) in the top right corner. Below the title bar, there is a text area that says: "Create a new entity of type: Business Service. When you have specified all required property values, and relationship targets, click 'Finish'." Below this, there are several sections with expandable headers (indicated by a downward arrow icon):

- Business Service Properties**: This section contains two fields:
 - Name:** A text box containing "Account Creation Eligibility".
 - Description:** A text box containing "Validate customer information is complete and can be verified before opening an account".
- Relationships**: This section is currently collapsed.
- Versions**: This section contains a link: "+ Add Capability Version".
- Charter**: This section is expanded and contains:
 - Name:** A text box containing "AccountCreationServiceCharter.doc". This text box is highlighted with a red rectangular border.
 - Replace Document**: A link with a pencil icon.
- Owning Organization**: This section contains a link: "+ Add Organization".

Figure 7-22 Adding a charter document

3. Assign an owning organization to the business service:
 - a. Still in the Create a Business service window, under the Owning Organization relationship, click **Add Organization**.
 - b. Enter C in the **Name** field, and select **Common Services** from the auto suggest list. The Common Services organization is added as a target of the owning organization relationship, as shown in Figure 7-23 on page 218.

Create a Business Service

Create a new entity of type: Business Service. When you have specified all required property values, and relationship targets, click 'Finish'.

Business Service Properties

Name: Account Creation Eligibility

Description: Validate customer information is complete and can be verified before opening an account

Relationships

Versions

+ Add Capability Version

Charter

Name: AccountCreationServiceCharter.doc

Replace Document

Owning Organization

Name: Common Services

Contact:

Contact Email:

Replace

Figure 7-23 Assigning an owning organization

- Click **Finish** to save your changes and create the business service. The displayed page changes to the Browse page. The Detail and Navigator widgets are updated to show the new business service, and the governance state, visible under Governance State, is Business Capability Identified, as shown in Figure 7-24 on page 219.

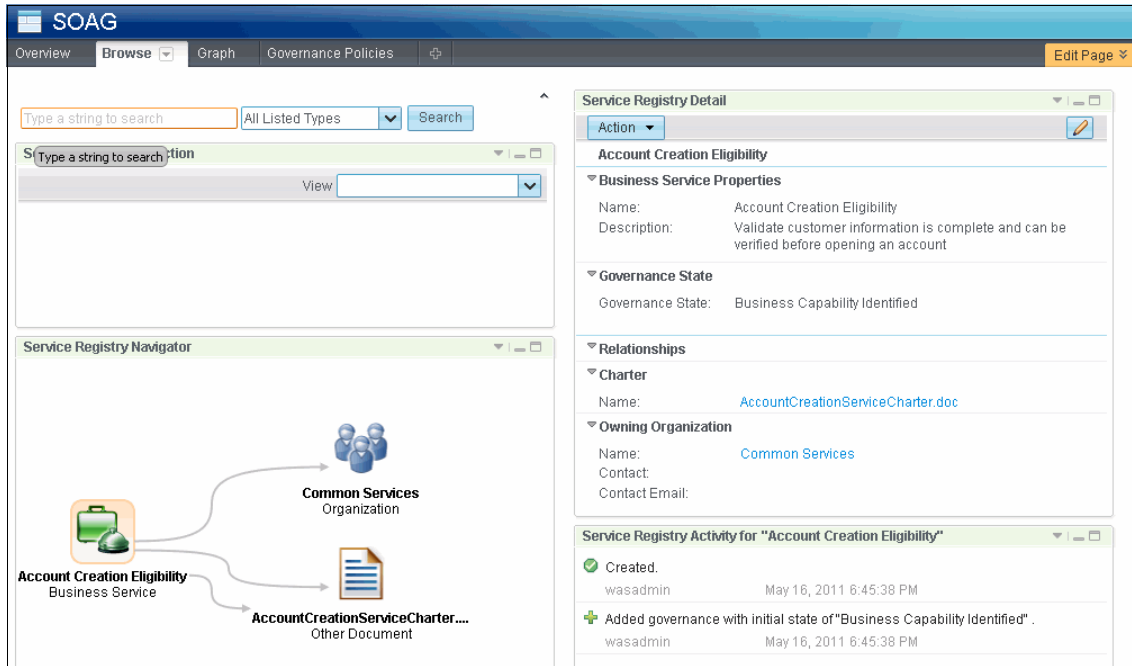


Figure 7-24 Account Creation Eligibility business service created

5. Move the Account Creation Eligibility business service through its life cycle to the Approved state:
 - a. In the Service Registry Detail widget, click **Action** → **Propose Charter**. In the Operation Successful window, click **OK**. The business service enters the *Charter Review* state, where an SOA Governance Center of Excellence can authorize or reject the capabilities, requirements, or ownership that are defined so far, as shown in Figure 7-25.

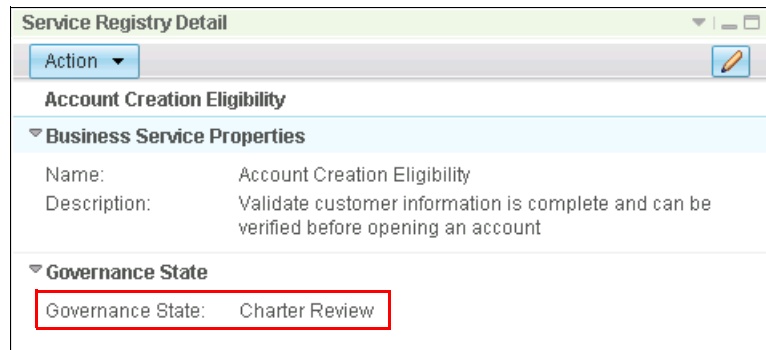


Figure 7-25 Governance state changes to Charter Review

- b. In the Service Registry Detail widget, click **Action** → **Approve Charter**. In the Operation Successful window, click **OK**. The business service enters the Business Capability Approved state, meaning that it becomes visible to any other potential users of the capability, as shown in Figure 7-26.

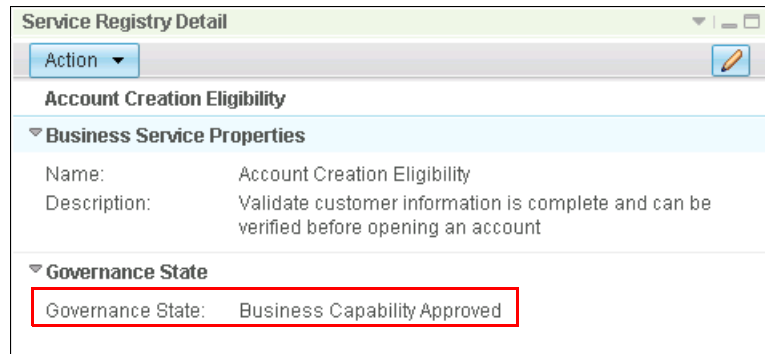


Figure 7-26 Governance state changes to Business Capability Approved

7.4.3 Registering a service

The next step in registering a new service is to load the WSDL that defines a service and associate it with a service version that is used to govern that service. You create a new service version called *Account Creation Service* and use it to initiate the service life cycle.

Loading the Account Creation Service into WSRR

You can use the Load Documents widget to load the WSDL for the supply company's Account Creation Service. You can also complete these steps directly in the WSRR web user interface.

To load the Account Creation Service into WSRR:

1. In the Browse page of the SOAG space, locate the Load Documents widget that you added to this page. Scroll down to the Load Documents section, as shown in Figure 7-27.

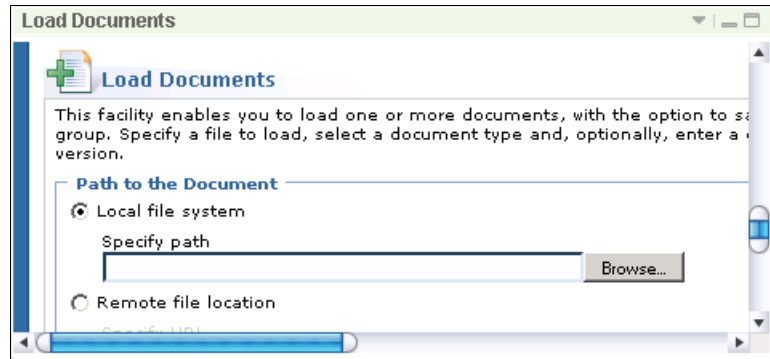


Figure 7-27 Locating the Load Documents section

The Account Creation Service is defined in the following files:

- AccountCreationInterfaceV1_0.wsdl
The interface WSDL file defining an AccountCreationV1_0 portType and createAccount operation.
- AccountCreationV1_0_DevelopmentPort.wsdl
The WSDL file defining the binding and service. This file imports the AccountCreationInterfaceV1_0.wsdl file.
- SupplyCompanyGlobalSchema.xsd
The XSD schema file defining complex types used by the Account Creation Service. This XSD file is imported by the AccountCreationInterfaceV1_0.wsdl file.

2. In the Load Documents widget, under Path to the Document, select **Local file system**, and browse to the AccountCreationInterfaceV1_0.wsdl file. Ensure that Document type is set to **WSDL**. Enter a description of AccountCreationInterface, and enter a document version of 1.0, as shown in Figure 7-28. Click **OK**.

Account Creation Service WSDL files: You can find the AccountCreationInterfaceV1_0.wsdl file and all the files that are required for the Account Creation Service in the additional material that is supplied with this book in the \Register_Service directory.

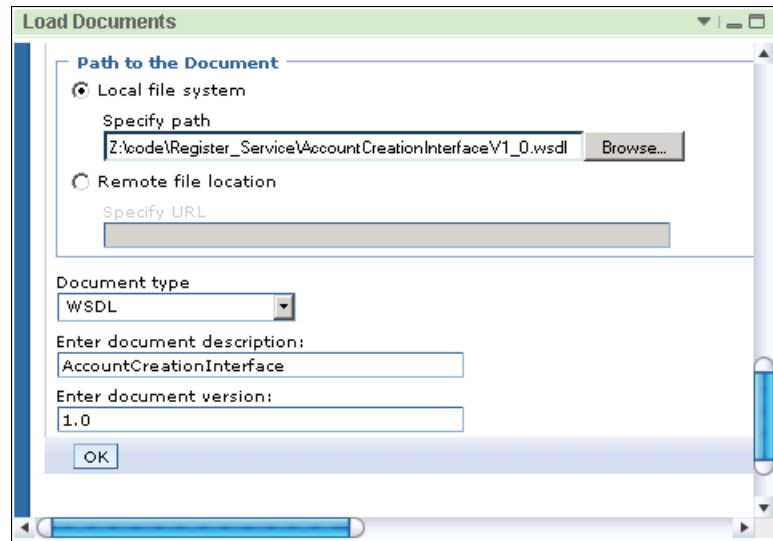


Figure 7-28 Loading AccountCreationInterfaceV1_0.wsdl

3. WSRR inspects the AccountCreationInterfaceV1_0.wsdl file and notices that the file imports the SupplyCompanyGlobalSchema.xsd XSD document, as shown in Figure 7-29. Click **Add**.



Figure 7-29 XSD file required

4. Browse to the `SupplyCompanyGlobalSchema.xsd` file. Set the Document type to XSD, enter a description of Supply Company Global Schema, and enter a version of 1.0. Click **OK**. Now, both documents are listed in the Documents to be Loaded section, as shown in Figure 7-30.

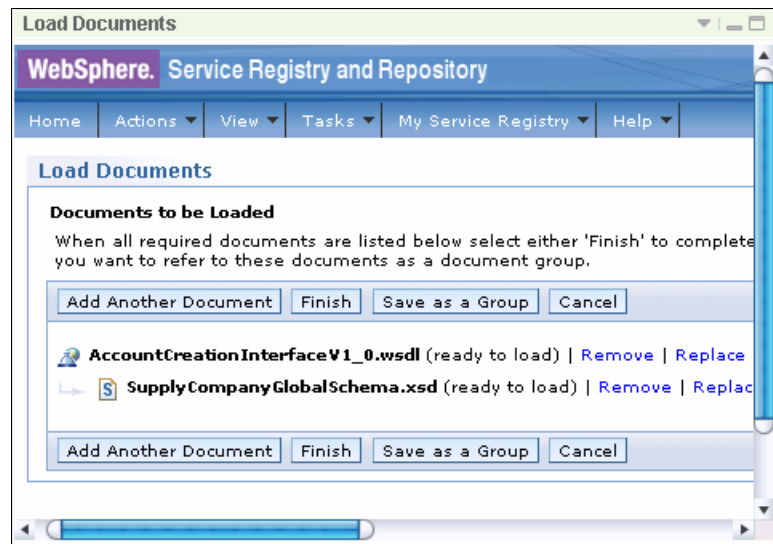


Figure 7-30 Loading `SupplyCompanyGlobalSchema.xsd`

5. Click **Add Another Document**. Load the AccountCreationV1_0_DevelopmentPort.wsdl file. Enter a document type of WSDL, a description of AccountCreation_DevelopmentPort, and a version of 1.0, then click **OK**. Three documents are now ready to be loaded, as shown in Figure 7-31.

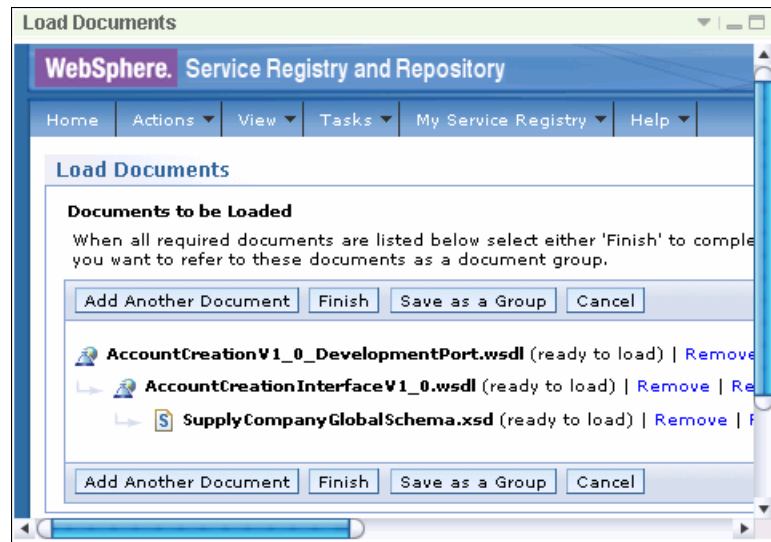


Figure 7-31 Loading AccountCreationV1_0_DevelopmentPort.wsdl

6. Click **Finish**. All three documents defining the Account Creation Service will be loaded into WSRR.

Creating a new service version

To add a service version:

1. In the SOAG space, click **Overview** to view the Overview page. The Approved Business Capabilities widget shows the Account Creation Eligibility business service, as shown in Figure 7-32. Click **Account Creation Eligibility**.



Figure 7-32 Approved Business Capabilities widget

2. The SOAG space moves to the Browse page, and the Detail, Navigator, and Activity widgets are updated to show the business service.
3. In the Service Registry Detail widget, click the **Edit the metadata for this object** icon, as shown in Figure 7-33.

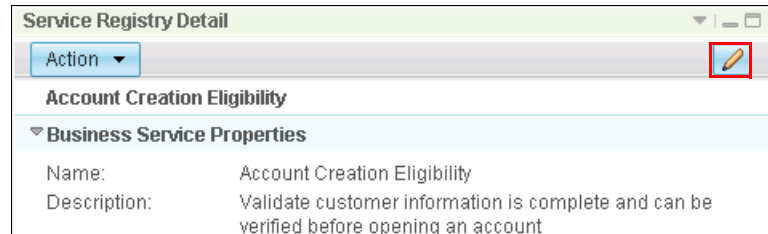


Figure 7-33 Edit the metadata for this object icon

4. In the Edit: Account Creation Eligibility window, click **Add Capability Version**. Select **Service Version** from the list, as shown in Figure 7-34, and click **Create**.

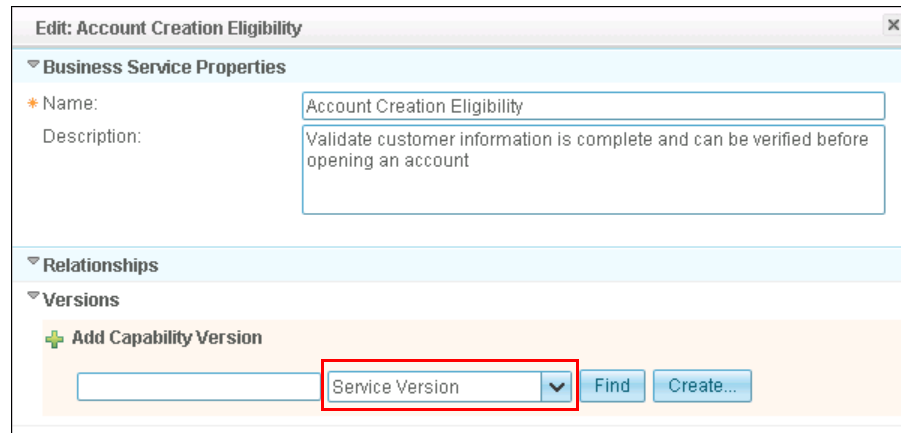


Figure 7-34 Selecting Service Version

5. In the Create: Service Version window, enter the following information as shown in Figure 7-35:
- Name: Account Creation Service
 - Description: Service to create new customer account
 - Version: 1.0.
 - Version Availability Date: Set to today's date
 - Version Termination Date: Set to one year from today's date

Create: Service Version

Create a new entity of type: Service Version. When you have specified all required property values, and relationship targets, click 'Finish'.

▼ Service Version Properties

★ Name: Account Creation Service

Description: Service to create new customer account

Namespace:

Version: 1.0

Consumer Identifier:

Version Availability Date: Monday, May 16, 2011

Version Termination Date: Wednesday, May 16, 2012

Figure 7-35 Creating a service version

6. Identify the service that is defined by the WSDL:
- Under the Provided Web Services relationship, click **Add Service**.
 - Enter an asterisk (*) in the name field, and click **Find**.
 - In the Select Targets for Provided Web Services window, select **AccountCreationV1_0(1.0)**, and click **Finish**.

The AccountCreationV1_0(1.0) Web service is added to the service version, as shown in Figure 7-36.

Create: Service Version

Create a new entity of type: Service Version. When you have specified all required property values, and relationship targets, click 'Finish'.

Service Version Properties

* Name: Account Creation Service

Description: Service to create new customer account

Namespace:

Version: 1.0

Consumer Identifier:

Version Availability Date: Monday, May 16, 2011

Version Termination Date: Wednesday, May 16, 2012

Relationships

Interface Specifications

+ Add Service Interface Specification

Provided Web Services

Name	Governance State
AccountCreationV1_0 (1.0)	

+ Add Service

Figure 7-36 Adding the provided web service

7. Assign an owning organization to the service version:
 - a. Under the Owning Organization relationship, click **Add Organization**.
 - b. Enter C in the name field, and select **Common Services**. The Common Services organization is added as a target of the owning organization relationship, as shown in Figure 7-37.

Owning Organization

Name: Common Services

Contact:

Contact Email:

Replace

Figure 7-37 Adding the owning organization

8. In the Create: Service Version window, click **Finish** to save your changes. The service version is created and added as target of the Capability Version relationship.
9. In the Edit: Account Creation Eligibility window, click **Finish** to save your changes. The Detail, Navigator, and Activity widgets are updated to show the modified business service.

Figure 7-38 shows the Service Registry Navigator widget.

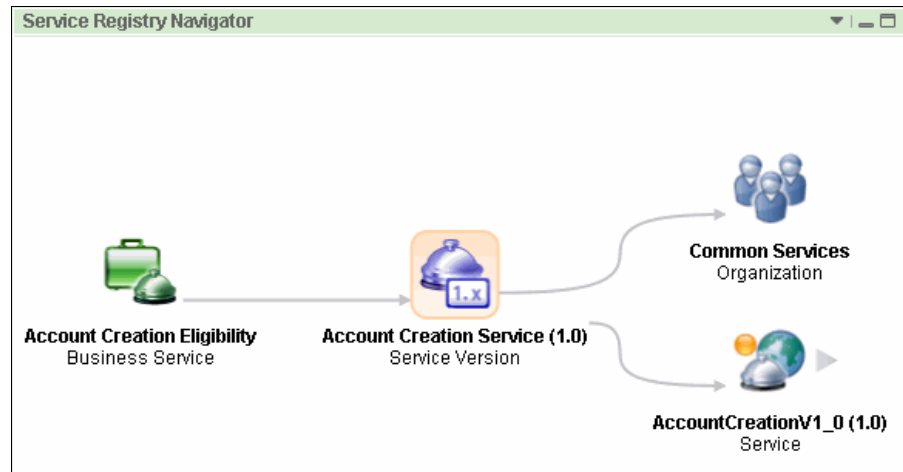
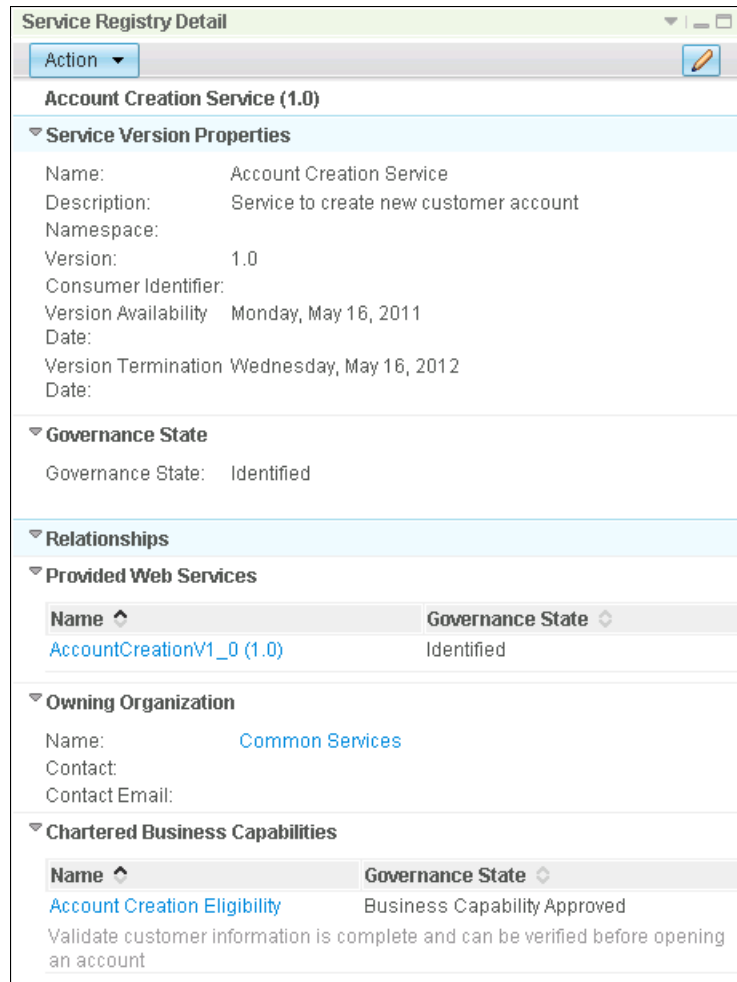


Figure 7-38 Service Registry Navigator widget

Figure 7-39 shows the Service Registry Detail widget.



The screenshot displays the 'Service Registry Detail' widget. At the top, there is a title bar with the text 'Service Registry Detail' and a small icon. Below the title bar is a tab labeled 'Action' with a pencil icon. The main content area is titled 'Account Creation Service (1.0)'. It contains several sections: 'Service Version Properties' with fields for Name, Description, Namespace, Version, Consumer Identifier, Version Availability, Date, and Version Termination; 'Governance State' with a field for Governance State; 'Relationships' with a sub-section 'Provided Web Services' containing a table with columns 'Name' and 'Governance State'; 'Owning Organization' with fields for Name, Contact, and Contact Email; and 'Chartered Business Capabilities' with a table containing columns 'Name' and 'Governance State'.

Service Version Properties	
Name:	Account Creation Service
Description:	Service to create new customer account
Namespace:	
Version:	1.0
Consumer Identifier:	
Version Availability	Monday, May 16, 2011
Date:	
Version Termination	Wednesday, May 16, 2012
Date:	

Governance State	
Governance State:	Identified

Relationships	
Provided Web Services	
Name	Governance State
AccountCreationV1_0 (1.0)	Identified

Owning Organization	
Name:	Common Services
Contact:	
Contact Email:	

Chartered Business Capabilities	
Name	Governance State
Account Creation Eligibility	Business Capability Approved
Validate customer information is complete and can be verified before opening an account	

Figure 7-39 Service Registry Detail widget

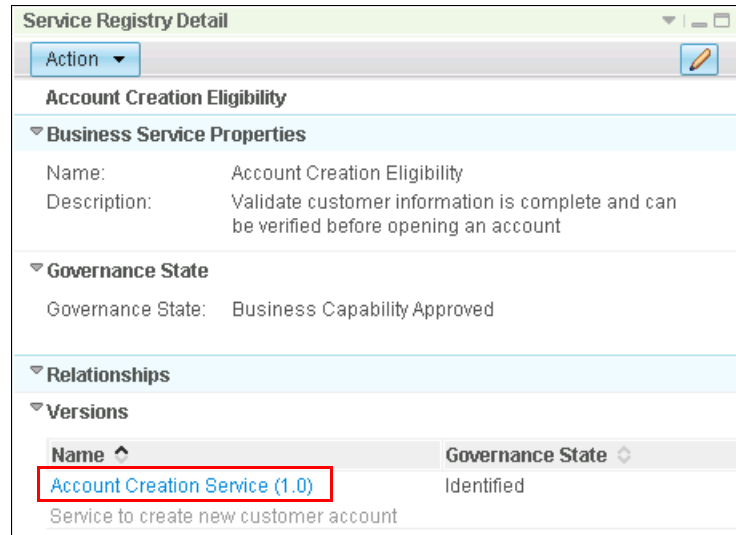
7.4.4 Providing scope and planning information for a service

After defining and approving the Account Creation Eligibility business service, and relating it to a service version, you must identify other planning and governance information for the service version.

First, define and agree the Account Creation Eligibility business service scope. This action ensures that the specific functional requirements and ownership for this version of the service are documented.

To provide scope and planning information for a service:

1. In the SOAG space, click **Overview** to return to the Overview page.
2. In the Approved Business Capabilities widget, click **Account Creation Eligibility** to display the business service details. The displayed page changes to the Browse page, and the Service Registry Detail widget shows the details of the business service.
3. In the Service Registry Detail widget, under Versions, click **Account Creation Service (1.0)**, as shown in Figure 7-40.



Service Registry Detail

Action

Account Creation Eligibility

Business Service Properties

Name: Account Creation Eligibility

Description: Validate customer information is complete and can be verified before opening an account

Governance State

Governance State: Business Capability Approved

Relationships

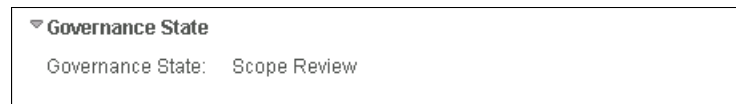
Versions

Name	Governance State
Account Creation Service (1.0)	Identified

Service to create new customer account

Figure 7-40 Service Registry Detail showing a business service

4. In the Service Registry Detail widget, click **Action** → **Propose Scope**. Click **OK** in the Operation Successful window. Note that the new governance state is Scope Review (Figure 7-41) and that Approve Scope and Revise Scope actions are now available in the Action menu.



Governance State

Governance State: Scope Review

Figure 7-41 Governance state of Scope Review

5. In the Service Registry Detail widget, click **Action** → **Approve Scope**. Click **OK** in the Operation Successful window. Note that the new governance state is Scoped (Figure 7-42) and that a Propose Plan action is now available in the Action menu.



Figure 7-42 Governance state of Scoped

You have already defined a Version Availability Date and Version Termination Date for the Account Creation Eligibility business service. Now, you approve the business service planning information to ensure that any dependencies are agreed and that availability and costing information is provided.

6. In the Service Registry Detail widget, click **Action** → **Propose Plan**. Click **OK** in the Operation Successful window. Note that the new governance state is Plan Review (Figure 7-43) and that Approve Plan and Revise Plan transitions are now available in the Action menu.

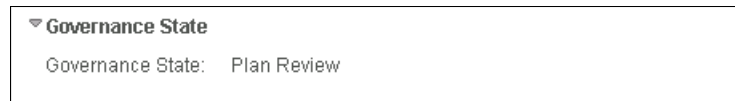


Figure 7-43 Governance state of Plan Review

7. In the Service Registry Detail widget click **Action** → **Approve Plan**. Click **OK** in the Operation Successful window. Note that the new governance state is Planned (Figure 7-44) and that a Propose Specification transition is now available in the Action menu.



Figure 7-44 Governance state of Planned

At this stage of the life cycle, the detailed consumption specification of the service starts.

7.4.5 Providing a specification and service level definition for a service

After completing the scoping and planning information, you must provide a clear statement about the service levels that consumers of this service can access. The service level definition (SLD) provides access to key information classes that are extracted from the WSDL when it is loaded.

To provide a specification and SLD for a service:

1. Create the SLD for the Account Creation Eligibility business service. This action encapsulates the endpoints, protocols, and interfaces that consumers can use to access the service.
 - a. In the SOAG space, click **Overview** to move to the Overview page.
 - b. In the Watch List - Service Registry Collection widget, click **Account Creation Service (1.0)** to open the Browse page.
 - c. The Service Registry Detail widget shows details about the Account Creation Service service version. Click the **Edit the metadata for this object** icon to edit these details.
 - d. Under the Provides relationship, click **Add Service Level Definition**. Click **Create**.
 - e. In the Create: Service Level Definition window, enter a Name of SLD - Account Creation Service, as shown in Figure 7-45.

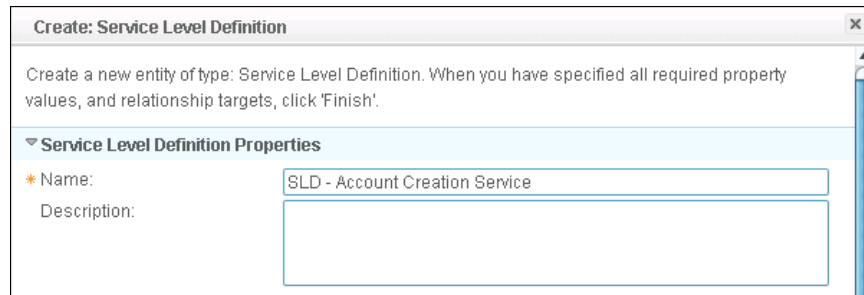


Figure 7-45 Defining an SLD

2. Define the service interface and the functional specification that are used to interact with the SLD:
 - a. In the Create: Service Level Definition window, click **Add Service Interface**.
 - b. Enter * in the Name field, and click **Find**.

- c. In the Select Targets for Service Interface window, select **AccountCreationV1_0**, and click **Finish**. The AccountCreationV1_0 interface is added as the service interface, as shown in Figure 7-46.

Create: Service Level Definition

Create a new entity of type: Service Level Definition. When you have specified all required property values, and relationship targets, click 'Finish'.

▼ Service Level Definition Properties

* Name: SLD - Account Creation Service

Description:

▼ Relationships

▼ Service Interface

Name: AccountCreationV1_0

Governance State:

Figure 7-46 Defining a service interface for the SLD

- d. Click **Finish** in the Create: Service Level Definition window.
- e. Click **Finish** in the Edit: Account Creation Service window.

3. Scope the SLD to provide the opportunity to define specific requirements or features that must be met. Customized SLDs can have many properties that must be set at this point. Follow these steps:
 - a. In the Service Registry Detail widget, scroll down and click the newly defined **SLD - Account Creation Service**. The Service Registry Detail widget now shows details about the SLD.
 - b. In the Service Registry Detail widget, click **Action** → **Propose Scope**. Click **OK** in the Operation Successful window. Note the new governance state is SLD Scope Review (Figure 7-47) and that Approve Scope and Revise Scope buttons are now displayed.

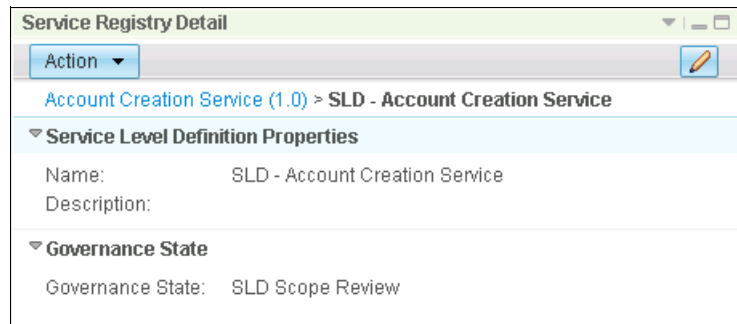


Figure 7-47 Governance State of SLD Scope Review

- c. In the Service Registry Detail widget, click **Action** → **Approve Scope**. Click **OK** in the Operation Successful window. Note that the new governance state is SLD Scoped (Figure 7-48) and that a Propose Specification action is now available in the Action menu.

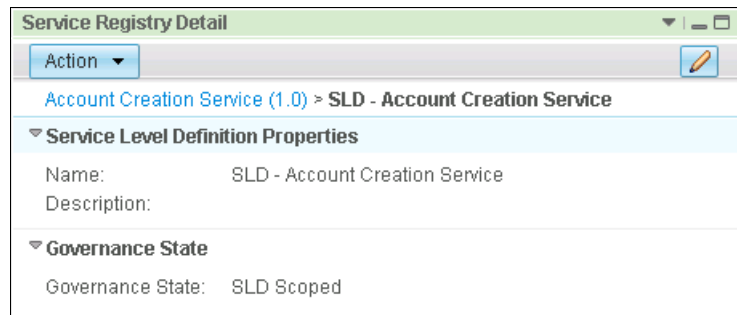


Figure 7-48 Governance State of SLD Scoped

4. Define the named port and the binding protocol used to access the SLD:
 - a. In the Service Registry Detail widget, click the **Edit the metadata for this object** icon to edit the SLA details.
 - b. In the Edit: SLD - Account Create Service window, click **Add Service Port** alongside the Bound Web Service Port relationship.
 - c. Enter * in the Name field, and click **Find**.
 - d. Select **AccountCreationServiceV1_0_DevelopmentPort (1.0)**, and click **Finish**. The Account Creation Service port is added as the bound web service port of the SLD, as shown in Figure 7-49.



▼ Bound Web Service Ports	
Name	Governance State
AccountCreationServiceV1_0_DevelopmentPort (1.0)	Planned

Figure 7-49 Bound web service port added to the SLD

5. Define the available endpoint that is currently deployed and that can be reached for this SLD:
 - a. Click **Add Service Endpoint** alongside the Available Endpoints relationship.
 - b. Enter * in the Name field, and click **Find**.
 - c. Select the SOAP service endpoint for the Account Creation Service, and click **Finish**. The Account Creation Service endpoint is added as the available endpoint of the SLD, as shown in Figure 7-50.
 - d. Click **Finish**.



▼ Available Endpoints	
Name	Governance State
https://localhost:9443/AccountCreationV1_0/services/AccountCreationServiceV1_0_DevelopmentPort (1.0)	Online

[Add Service Endpoint](#)

Figure 7-50 Available endpoint added to the SLD

6. Complete the SLD specification:
 - a. In the Service Registry Detail widget, click **Action** → **Propose Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is SLD Review (Figure 7-51) and that Approve Specification and Revise Specification are now available in the Action menu.

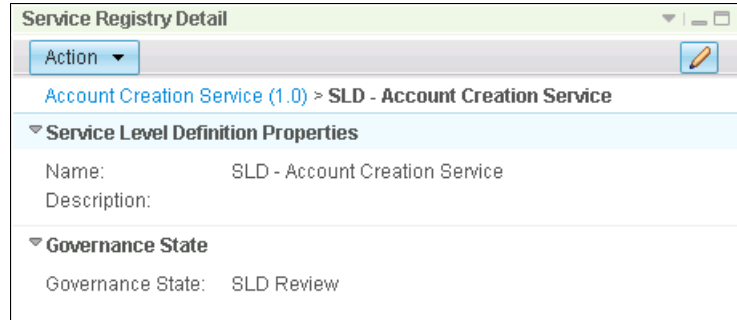


Figure 7-51 Governance State of SLD Review

- b. In the Service Registry Detail widget, click **Action** → **Approve Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is SLD Subscribable (Figure 7-52) and that Supersede and Deprecate actions are now available in the Action menu. This SLD is subscribable, meaning that SLAs can now be requested against it.

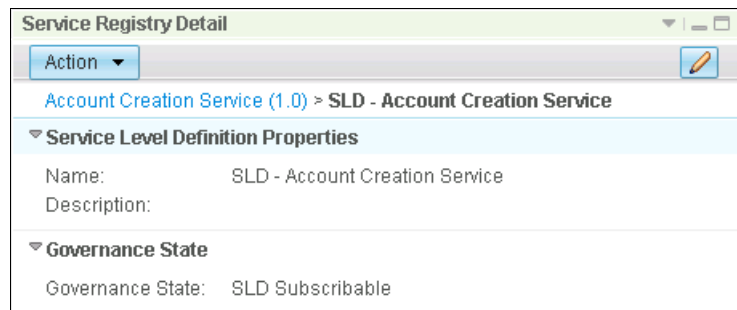


Figure 7-52 Governance State of SLD Subscribable

7. Complete the Account Creation service specification:
 - a. Click **Overview** to return to the Overview page.
 - b. In the Watch List - Service Registry Collection widget, click **Account Creation Service (1.0)**. The displayed page changes to the Browse page,

and the details of the Account Creation Service are shown in the Service Registry Detail widget.

- c. In the Service Registry Detail widget, click **Action** → **Propose Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is Specification Review (Figure 7-53) and that Approve Specification and Revise Specification buttons are now available in the Action menu.

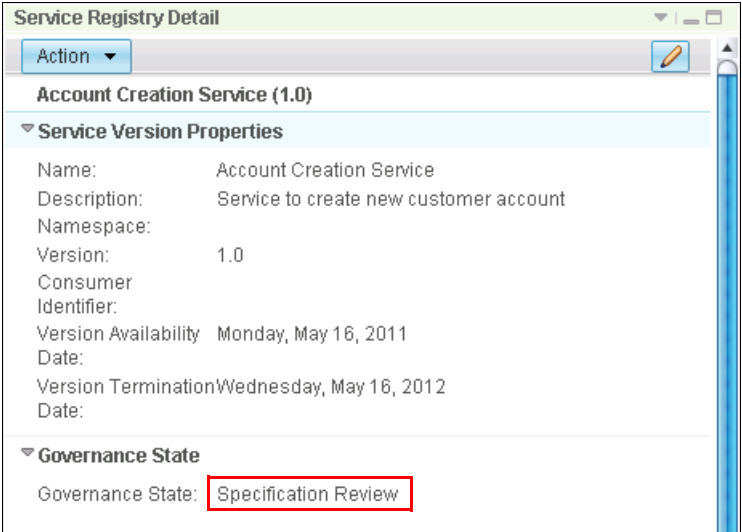


Figure 7-53 Governance State of Specification Review

- d. In the Service Registry Detail widget, click **Action** → **Approve Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is Specified (Figure 7-54) and that a Propose Realization action is now available in the Action menu.



Figure 7-54 Governance State of Specified

8. Complete the Account Creation Service realization. Because the service is already deployed, you tie in any deployed or installable assets during this next stage. Follow these steps:
 - a. Click **Action** → **Propose Realization**. Click **OK** in the Operation Successful window. Note that the new governance state is Realization Review (Figure 7-55) and that Approve Realization and Revise Realization buttons are now available in the Action menu.

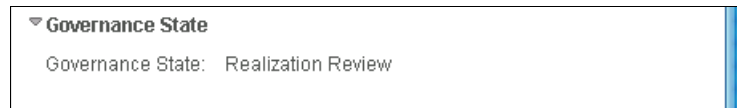


Figure 7-55 Governance State of Realization Review

- b. Click **Action** → **Approve Realization**. Click **OK** in the Operation Successful window. Note that the new governance state is Realized (Figure 7-56) and that a Propose Staging Deployment button is now available in the Action menu.

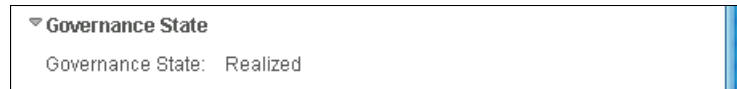


Figure 7-56 Governance State of Realized

7.4.6 Governing the service consumer

In this section we create an SLA between the service consumer, Account Management Consumer, and the service provider, Account Creation Service. This section contains the following sections:

- ▶ Identifying a requirement for a new business capability
- ▶ Scoping an application version
- ▶ Creating a service level agreement

Identifying a requirement for a new business capability

An important part of scoping an application is to identify how it adds value to the business. WSRR uses business capabilities to define the business view of an application or service.

If there is no existing business capability that the service realizes, then you must create and define a business application. You need to add the following key defining components:

- ▶ A service charter to define the scope and role of the business application
- ▶ An owning organization

The owning organization is responsible for defining requirements for the capability, and owning any realizations of this capability.

To identify a requirement:

1. Create a business application:
 - a. In the SOAG space, click **Overview** to view the Overview page.
 - b. In the Service Registry Actions widget, click **Create a Business Application**.
 - c. In the Create a Business Application window, enter the name: Account Management Portal.
 - d. Under Charter, click **Add Other Document**. Click **Browse**, and navigate to AccountManagementPortalCharter.doc. Click **Load**.

AccountManagementPortalCharter.doc: You can find the AccountManagementPortalCharter.doc file in the additional material that is supplied with this book in the \Register_Service directory.

- e. The document is added to the charter of the Account Management Portal business application, as shown in Figure 7-57. Click **Finish**.

Create a Business Application

Create a new entity of type: Business Application. When you have specified all required property values, and relationship targets, click 'Finish'.

▼ **Business Application Properties**

* Name: Account Management Portal

Description:

▼ **Relationships**

▼ **Versions**

+ Add Capability Version

▼ **Charter**

Name: AccountManagementPortalCharter.doc

Replace Document

Figure 7-57 Creating the Account Management Portal business application

2. Move the Account Management Portal application through its life cycle to the Approved state:
- a. In the Service Registry Detail widget, click **Action** → **Propose Charter**. Click **OK** in the Operation Successful window. The business application enters the Charter Review state (Figure 7-58), where an SOA Governance Center of Excellence can authorize or reject the capabilities, requirements, or ownership that are defined so far.

Service Registry Detail

Action

Account Management Portal

▼ **Business Application Properties**

Name: Account Management Portal

Description:

▼ **Governance State**

Governance State: Charter Review

Figure 7-58 Governance State of Charter Review

- b. In the Service Registry Detail widget, click **Edit the metadata for this object**.
- c. In the Edit: Account Management Portal window, click **Add Organization**.
- d. Enter C in the Name field, and select **Common Services** from the list. The Common Services organization is added as a target of the Owning organization relationship. Click **Finish**.
- e. In the Service Registry Detail widget, click **Action** → **Approve Charter**. Click **OK** in the Operation Successful window. Note that the new governance state is Business Capability Approved (Figure 7-59), meaning that it becomes visible to any other potential users of the capability.

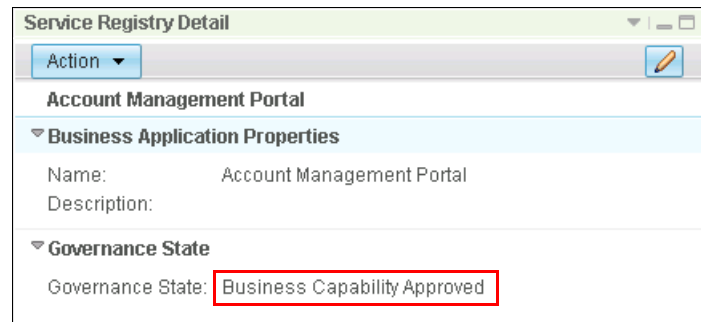


Figure 7-59 Governance State of Business Capability Approved

Creating a capability version

1. Reference the Account Management Consumer capability version from the Account Management Portal business application:
 - a. In the Service Registry Detail widget, click **Edit the metadata for this object**.
 - b. In the Edit: Account Management Portal window, click **Add Capability Version**. Click **Create** to create a new capability version.

- c. Enter the name: Account Management Consumer and click **Finish**. An Account Management Consumer capability version is added to the Account Management Portal business application, as shown in Figure 7-60.

Edit: Account Management Portal	
Business Application Properties	
Name:	Account Management Portal
Description:	
Relationships	
Versions	
Name	Governance State
Account Management Consumer	Identified

Figure 7-60 Adding a capability version

- d. Click **Finish**.

Scoping an application version

An application version in the governance enablement profile represents a specific version or release of an application and provides a range of functional and non-functional specifications for that version of the application. These specifications are tied to a specific service version from the provider but are referenced through the application version of the consumer.

Assigning a consumer identifier to the application version

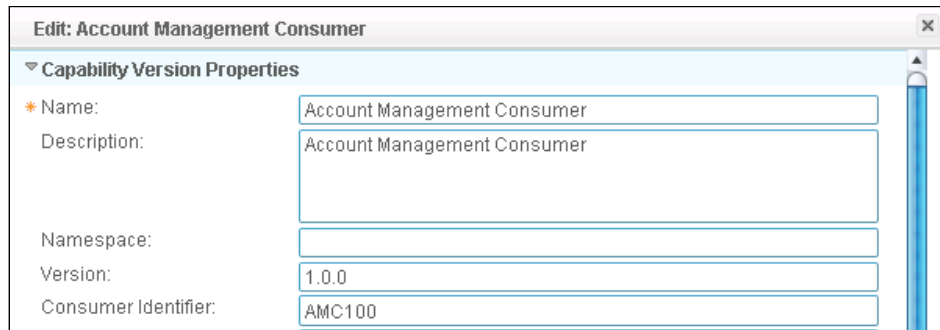
After the business capability is defined, reviewed, and approved, it is the responsibility primarily of the development organization to create an implementation of the application. However, in this example, the application version is realized through an agreement with another department. In this respect, it is important to assign the application version a unique consumer identifier so that the provider can tie in the correct consumer that is accessing the service.

To assign a consumer identifier:

1. The Service Registry Detail widget shows details of the Account Management Portal business application. Under Versions, click **Account Management Consumer** to display information about this capability version.
2. In the Service Registry Detail widget, click **Edit the metadata for this object**.

3. In the Edit: Account Management Consumer window, set the following information, as shown in Figure 7-61:
 - Description: Account Management Consumer
 - Version: 1.0.0
 - Consumer Identifier: AMC100

Consumer Identifier: The Consumer Identifier is an identifier that the application passes in the header of all service invocations it attempts. The invoked services use this identifier to determine whether the consumer has the appropriate service level agreements in place to make the invocation. The format of the Consumer Identifier is defined by your enterprise.



The screenshot shows a window titled "Edit: Account Management Consumer". Inside, there is a section titled "▼ Capability Version Properties". Below this section are five labeled text input fields:

- Name:** Account Management Consumer
- Description:** Account Management Consumer
- Namespace:** (empty)
- Version:** 1.0.0
- Consumer Identifier:** AMC100

Figure 7-61 Setting the capability version properties

4. Under Owning Organization, click **Add Organization**.
5. Enter C in the Name field, and select **Common Services** from the list. The Common Services organization is added as a target of the Owning organization relationship.
6. Click **Finish**.

Proposing and approving the application version scope

Now that the scope of the application version is defined, it must be put out for review so that all potential consumers of the application can verify that their requirements are within the proposed scope by the development team.

To propose and approve the application version scope:

1. In the Service Registry Detail widget, click **Action** → **Propose Scope**. Click **OK** in the Operation Successful window. Note that the new governance state is Scope Review (Figure 7-62).

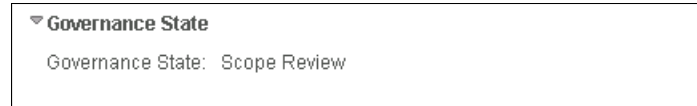


Figure 7-62 Governance Scope of Scope Review

In the Scope Review state, the SOA governance team reviews the application version requirements and carries out the following verifications:

- This application version is warranted across the organization.
- The requirements and stakeholders have been agreed.
- The owning organization, responsible for delivering the requirements, has been identified and assigned to the application version.

When the service version scope review is complete, the scope is approved.

2. In the Service Registry Detail widget, click **Action** → **Approve Scope**. Click **OK** in the Operation Successful window. Note that the new governance state is Scoped (Figure 7-63).



Figure 7-63 Governance State of Scoped

Creating a service level agreement

We use the governance enablement profile to create a new service level agreement (SLA) and to submit and approve an SLA request. In this example, the SLA is between the consumer, Account Management Consumer, and the provider, Account Creation Service.

To create an SLA:

1. The Service Registry Detail widget shows details of the Account Management Consumer capability version. In the Service Registry Detail widget, click **Edit the metadata for this object**.
2. In the Edit: Account Management Consumer window, under **Consumes**, click **Add Service Level Agreement**. Click **Create**.

3. In the Create: Service Level Agreement window, enter the following settings, as shown in Figure 7-64:
 - Name: AMC100 Account Creation Service SLA
 - Context Identifier: AMC100CI01

Context Identifier: This setting allows the consumer to identify which SLA is in play within a particular relationship with the provider if there are more than one. The naming convention for Context Identifiers is specific to your own enterprise.

- Subscription Availability Date: Set to today's date
- Subscription Termination Date: Set to one year from today's date

Create: Service Level Agreement

Create a new entity of type: Service Level Agreement. When you have specified all required property values, and relationship targets, click 'Finish'.

▼ Service Level Agreement Properties

* Name: AMC100 Account Creation Service SLA

Description:

Context Identifier: AMC100CI01

Subscription Availability Date: Monday, May 16, 2011

Subscription Termination Date: Wednesday, May 16, 2012

Version Match Criteria: LatestCompatibleVersion

Figure 7-64 Defining the SLA

4. Associate the SLA with the SLD. Under Agreed Endpoints, click **Add Service Level Definition**.
5. Enter an asterisk (*) in the Name field, and select **SLD - Account Creation Service** from the list. The service level definition is now associated with the SLA, as shown in Figure 7-65.

▼ Relationships	
▼ Agreed Endpoints	
Name ↕	Governance State ↕
SLD - Account Creation Service	SLD Subscribable

Figure 7-65 Associating the service level definition with the SLA

6. In the Create: Service Level Agreement window, click **Finish**.
7. In the Edit: Account Management Consumer window, click **Finish**.

The next step is to request the SLA. By requesting the SLA, the development team is effectively asking the provider of the service for permission to use the service level definition associated with the application version.
8. In the Service Registry Detail widget, under Consumes, click **AMC100 Account Creation Service SLA** to show details about the SLA.
9. In the Service Registry Detail widget, click **Action** → **Request SLA**. Click **OK** in the Operation Successful window. Note that the new governance state is SLA Requested (Figure 7-66).

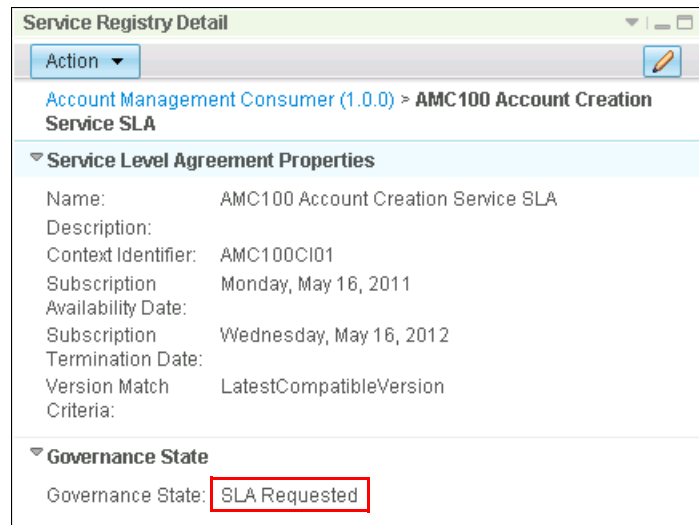


Figure 7-66 Governance State of SLA Requested

The provider of the service has the option to approve or reject the request, or ask for it to be revised. Here, the request is approved, which moves it to the Inactive state. Thus, the development team that wants to consume the service can continue development based on the consumption of this specific SLD, but they do not yet have authorization to access any endpoints.

10. In the Service Registry Detail widget, click **Action** → **Approve SLA Request**. Click **OK** in the Operation Successful window. Note that the new governance state is SLA Inactive (Figure 7-67).



Figure 7-67 Governance State of SLA Inactive

Now that the service version has been defined and the service level definition is in place, the service version can be submitted to be reviewed and approved by the SOA Governance team.

11. Before we activate the SLA, move the Account Management Consumer into the Specified state.
 - a. In the Service Registry Detail click **Account Management Consumer (1.0.0)** to show the details for the consumer (Figure 7-68). Note that Account Management Consumer is currently in the Staged state.



Figure 7-68 Account Management Consumer (1.0.0) link

- b. In the Service Registry Detail widget, click **Action** → **Propose Plan**. Click **OK** in the Operation Successful window. Note that the new governance state is Plan Review.
 - c. In the Service Registry Detail widget, click **Action** → **Approve Plan**. Click **OK** in the Operation Successful window. Note that the new governance state is Planned.
 - d. In the Service Registry Detail widget, click **Action** → **Propose Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is Specification Review.
 - e. In the Service Registry Detail widget, click **Action** → **Approve Specification**. Click **OK** in the Operation Successful window. Note that the new governance state is Specified (Figure 7-69 on page 248).

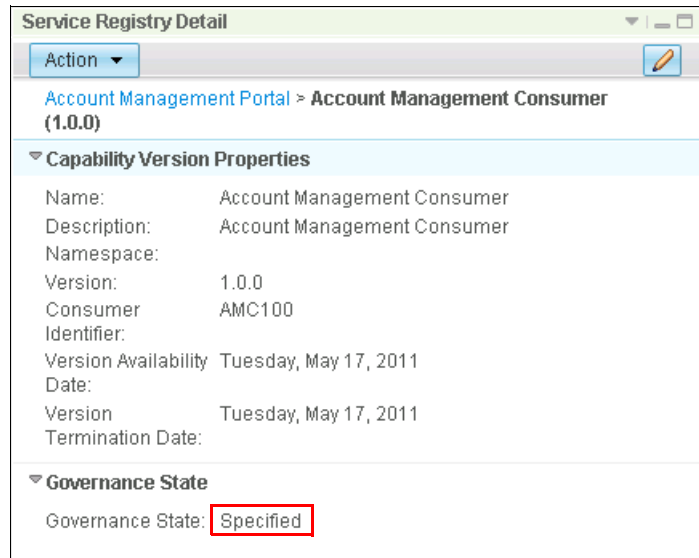


Figure 7-69 Governance State of Specified

Because there is a suitable endpoint available to invoke, the operations manager activates the SLA.

12. In the Service Registry Detail widget, click **AMC100 Account Creation Service SLA**.
13. In the Service Registry Detail widget, click **Action** → **Activate SLA**. Click **OK** in the Operation Successful window. Note that the new governance state is SLA Active (Figure 7-70).



Figure 7-70 Governance State of SLA Active

At this point, the Account Management Consumer (1.0.0) application is in Specified state. It has an active SLA to consume the Account Creation Service (1.0). Hence you can begin to develop and test the consumer application in the development environment.

7.4.7 Completing the services life cycle

After you have described the service in sufficient detail for it to be consumed and programmed against and have deployed an implementation of the service, you

must declare the endpoint in the registry so that it is promoted to, and therefore visible in, the appropriate environments. To complete the services life cycle:

1. In the SOAG space, click **Overview** to view the Overview page.
2. In the Approved Business Capabilities widget, click **Account Creation Eligibility**. This will open the Browse page.
3. In the Service Registry Detail page, under Versions click **Account Creation Service (1.0)**.
4. In the Service Registry Detail widget, click **Action** → **Propose Staging Deployment**. Click **OK** in the Operation Successful window. Note that the new governance state is Staging Review (Figure 7-71) and that Approve Staging Deployment and Revise Staging Deployment are now visible in the Actions menu. The operations team now has the opportunity to verify that the information in the governance registry is correct for deployment to the staging registry.

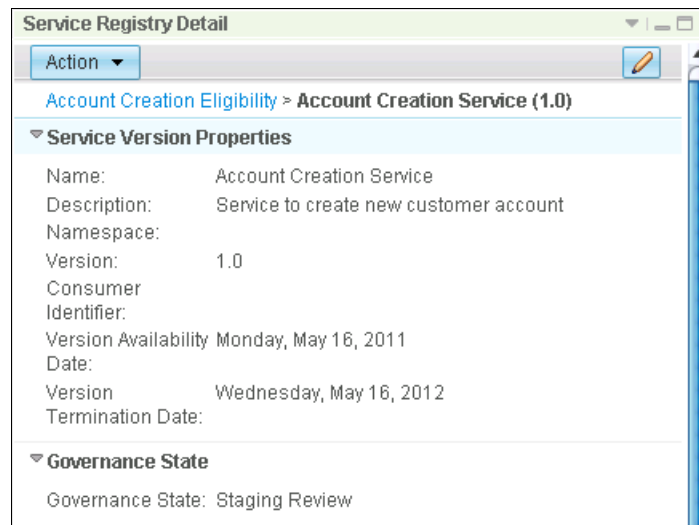


Figure 7-71 Governance State of Staging Review

5. In the Service Registry Detail widget, click **Action** → **Approve Staging Deployment**. Click **OK** in the Operation Successful window. Note that the new governance state is Staged (Figure 7-72).



Figure 7-72 Governance State of Staged

In the staging environment, testing of the Account Creation Service is carried out to ensure that if it is made available in production that it will meet all of its proposed SLDs and SLAs. We leave the service in the staging environment for now, but after testing is complete, the service can be proposed and approved for certification and deployed to a production environment.



Implementing a mediation

This chapter provides step-by-step instructions describing how to implement a mediation proxy module that is deployed in WebSphere Enterprise Service Bus (WebSphere ESB). The mediation module functions as a facade for any client that needs to invoke a service without having to bother with service information, such as the correct service endpoint or version. The mediation module proxy is implemented with basic capabilities to determine physical service endpoint destinations from WebSphere Service Registry and Repository (WSRR) and to check for existing service level agreements (SLAs).

In the first part of this chapter, we discuss the benefits of a mediation proxy module and introduce the basic mediation primitives to integrate WebSphere ESB with WSRR by providing the mediation with dynamic and flexible capabilities.

In the second part of this chapter, we describe step-by-step instructions to implement the mediation proxy module using the IBM Integration Designer and the WSRR Eclipse Plug-in.

8.1 Addressing the business requirements

When you build a mediation proxy module, you need to consider both the architectural decisions and implementation approaches to ensure that the following business requirements are met, as described in Chapter 5, “The case study and scenario used in this book” on page 133:

- ▶ Using WebSphere ESB provides a connectivity layer between the clients and the service provider. Implementing and placing a mediation module that functions as a proxy and facade between the clients and the service provider ensures that no point-to-point connections are used. This level of isolation and transparency meets the business requirement NFR-101.
- ▶ To address business requirements NFR-201 and NFR-202, the mediation module needs to be able to retrieve service endpoint information dynamically at run time. Introducing an Endpoint Lookup mediation primitive that connects to WSRR at run time achieves this requirement.
- ▶ Business requirement NFR-203 focuses more on security than the dynamic and flexible functions. The mediation needs to be able to query WSRR at run time for an existing contract between the caller and the service provider. Adding an SLA check that enforces a contract between the caller and the service provider at run time satisfies this business requirement.

8.2 Benefits of a mediation proxy module

In numerous existing scenarios, service consumers typically request a service provider from a variety of runtime environments, where the service information, such as the physical service endpoint of a service provider, is hard coded in the service consumer’s implementation.

Figure 8-1 shows this scenario in detail. The service consumers represented by the clients A, B, and C are running in separate run times and connect directly to the service provider that is represented by Service A. All clients are closely coupled to Service A by a point-to-point connection that is hard coded in their implementation.

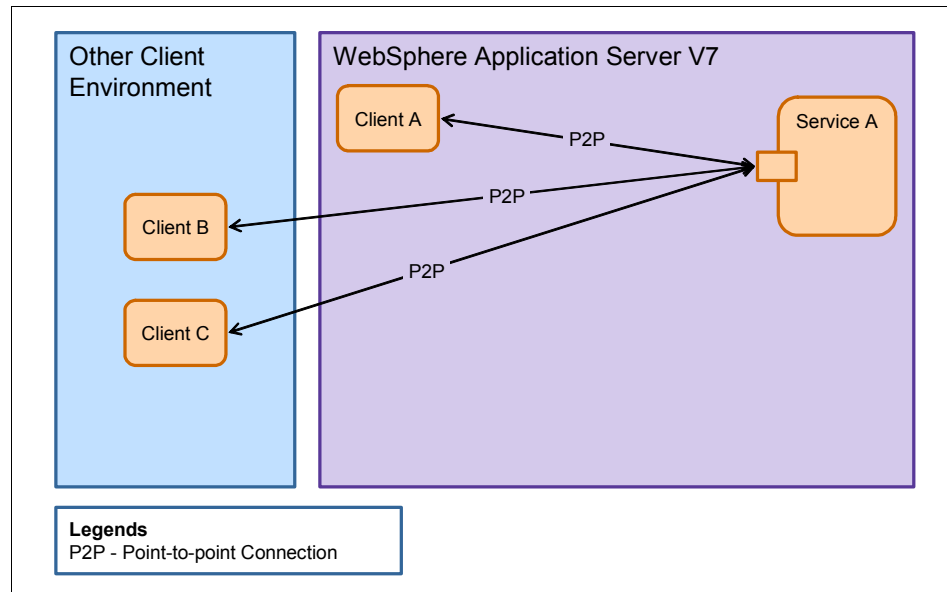


Figure 8-1 Point-to-point communication between service consumers and a service provider

This scenario has various challenges when implemented in rapidly changing requirements, such as operational issues when service versions change or when the service provider moves to another location. In addition, security-relevant issues cannot be addressed easily, for example when checking whether the client that connects has the right to request a specific service provider.

Introducing WebSphere ESB Registry Edition addresses the challenges of a closely-coupled service architecture. A fundamental component of WebSphere ESB Registry Edition, WebSphere ESB, provides a connectivity layer that enables flexible and simple integration of services, thereby decoupling integration logic from applications.

As shown in Figure 8-2, WebSphere ESB Registry Edition, respectively the mediation proxy module, is placed as a facade between clients A, B, and C and the service provider Service A, thereby isolating the clients and their implementation from the service provider. Although the WebSphere ESB mediation proxy module has decoupled the service consumer from the service provider, the mediation proxy still has point-to-point links with the service. When endpoints are not decoupled from mediation code, the same limitations exist as shown in Figure 8-1 on page 253.

For example, promotion to other development environments requires significant changes to the WebSphere ESB configuration and the mediation code. Therefore, the environment needs to use WSRR, which is the other crucial component of WebSphere ESB Registry Edition.

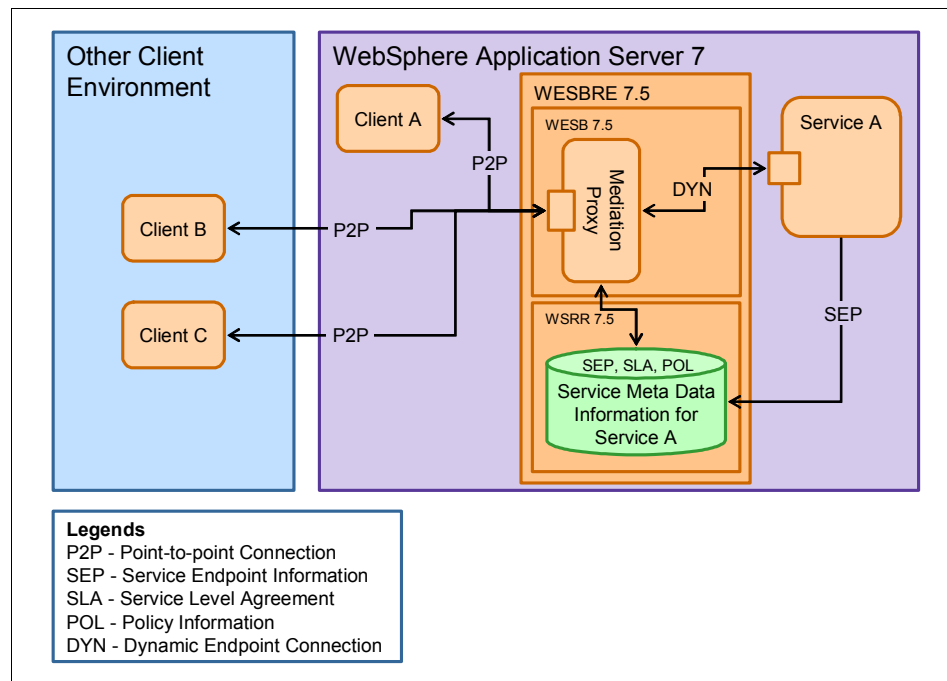


Figure 8-2 Mediation proxy module to isolate service consumer from the service provider

WSRR increases the agility of WebSphere ESB through the following mediation primitives:

- ▶ Endpoint Lookup
- ▶ SLA Check
- ▶ SLA Endpoint Lookup
- ▶ Policy Resolution
- ▶ Gateway Endpoint Lookup

These primitives are key to accelerating robust flexible mediations, which can be changed rapidly without redeployment of the mediation itself. We discuss the first two primitives in more detail in 8.3, “Basic primitives to integrate a smart WebSphere ESB Registry Edition solution” on page 256.

After WSRR is populated with service metadata information about the service provider Service A, the integration and use of the primitives mentioned previously within a WebSphere ESB mediation module provides more dynamic and flexible functions. A dynamic WebSphere ESB mediation uses WSRR to perform WebSphere ESB functions at run time based on the metadata that is stored in the registry. It can determine a back-end routing destination based on availability metrics, retrieve security policy data for policy based enforcement, and retrieve artifacts and mediation logic based on environmental data.

With the advanced mediation primitives, such as the Gateway Endpoint Lookup, the mediation proxy module can be enabled to function as a service router, retrieving any incoming service request from any service consumer and then routing to any service endpoint for any service provider. This routing functionality can be combined with the other advantages that the remaining primitives provide, such as SLA checks or policy resolution.

Note on this basic scenario: In this basic scenario the mediation proxy module focuses only on a specific service provider without the need for routing functionality.

8.2.1 Benefits of a dynamic endpoint lookup in a mediation

An SCA Import component in a mediation module defines an exit point for transmitting outbound service requests. SCA Import components in WebSphere ESB contain a hard coded endpoint URL that points to a specific service provider location. Hard coding the destination endpoint URL in the mediation logic has limitations. The primary issue is that any changes to the endpoint URL also require a change to the Import component in WebSphere ESB.

However, by using the Endpoint Lookup mediation primitive, the mediation module retrieves the destination location from a WSRR instance dynamically at run time. Dynamically retrieving the endpoint instead of hard coding it has a number of advantages:

- It is easier to move mediation code between environments. For example, development, test, and production endpoints for the same service are likely to be different. If the endpoint URL is not hard coded, the mediation code itself does not need to be changed when changes are promoted.

- ▶ Maintaining the list of endpoints in WSRR reduces the need for changes to the WebSphere ESB code and, in turn, lowers the time and cost that are required to make changes for service endpoints.
- ▶ Service endpoints can be updated in WSRR based on service availability. For example, if an endpoint is unavailable, WSRR can provide an available endpoint instead.

8.2.2 Benefits of an SLA check in a mediation

The WebSphere ESB Registry Edition mediation module implementations provide dynamic and flexible functions. However, users can be worried that they have no visibility into who is actually using the services that are defined in the registry. Introducing SLAs that define and track who uses a service addresses these concerns.

The benefit of using the SLAs that are defined in WSRR at run time is that the contracts identified by the SOA governance team can be enforced at run time. This identification ensures that the relationships modeled in WSRR represent the real state of the SOA connectivity infrastructure.

The governance enablement profile, described in Chapter 6, “Governance enablement profile” on page 143, provides an SLA entity to define relationships between consumer capabilities and services. In addition, WebSphere ESB provides the following mediation primitives to query and enforce these SLAs:

- ▶ SLA Check mediation primitive
- ▶ SLA Endpoint Lookup mediation primitive

8.3 Basic primitives to integrate a smart WebSphere ESB Registry Edition solution

This section provides a brief overview of the basic mediation primitives that can be used to achieve a dynamic and flexible mediation proxy module. Figure 8-3 shows the primitives that are fundamental to increase dynamicity and flexibility in a WebSphere ESB mediation module using WSRR. The Endpoint Lookup and SLA Check mediation primitives are considered the mediation primitives that provide basic functions, such as dynamic endpoint retrieval and validation for an existing SLA.



Figure 8-3 WESBRE primitives to integrate WebSphere ESB with WSRR

This section first discusses how to use the Endpoint Lookup mediation primitive to dynamically retrieve service endpoints. Later it describes the SLA Check primitive to validate whether a requesting service consumer has an active SLA with the requested service provider.

Advanced functions: The SLA Endpoint Lookup, Policy Resolution, and Gateway Endpoint Lookup mediation primitives provide advanced functions, such as security enforcement and routing. We discuss these primitives in more detail in Chapter 9, “Extending the mediation” on page 315.

8.3.1 Endpoint Lookup mediation primitive

The Endpoint Lookup mediation primitive is used to dynamically route a message to an existing service endpoint. It searches for service information in WSRR and retrieves either none, one, or all matching service endpoints. The service endpoint information that is retrieved can relate directly to web services, to SCA module exports, or to manually-defined services, such as HTTP or JMS services.

As shown in Figure 8-4, the Endpoint Lookup mediation primitive consists of one input terminal, one fail terminal (rectangle shaped terminal), and two output terminals (out and noMatch). The input terminal is wired to accept a message, and the other terminals are wired to propagate a message.



Figure 8-4 Endpoint Lookup mediation primitive

If service endpoints are retrieved successfully from WSRR, the output terminal (out) propagates the original incoming message, which is modified by the service endpoint information and possible alternate service endpoints. If no matching services are found in the registry, the output terminal (noMatch) propagates the unmodified original incoming message. If an exception occurs during the

processing of the incoming message, the fail terminal propagates the original incoming message and any exception information.

Implementing dynamic routing on request: For the run time to implement dynamic routing on a request, you must set the “Use dynamic endpoint if set in the message header” property in the Service Invoke mediation primitive or the Callout node. You must also set a default endpoint that the run time uses if no dynamic endpoint is found. See 8.3.3, “Service Invoke mediation primitive and Callout node” on page 262 for further details.

When the Endpoint Lookup mediation primitive receives a message, it sends a search query to WSRR. The search query is constructed using the Endpoint Lookup properties that are specified in the Details and Advanced tab in the Properties view. When a search query is successful, the Endpoint Lookup mediation primitive can make updates to both the message context in the `primitiveContext` element and to the message headers as follows:

- ▶ A dynamic service endpoint address for a callout or service invoke is stored in `/headers/SMOHeader/Target/address`
- ▶ A list of alternate service endpoint addresses is stored in `/headers/SMOHeader/AlternateTarget`
- ▶ More service information about the query results from the WSRR is stored in `/context/primitiveContext/EndpointLookupContext`

The properties settings for the Endpoint Lookup primitive query go beyond just looking for a particular port type. As shown in Figure 8-5, other attributes such as Namespace and Version are supported, as are custom properties and classifications. The Endpoint Lookup primitive also specifies a Match Policy.

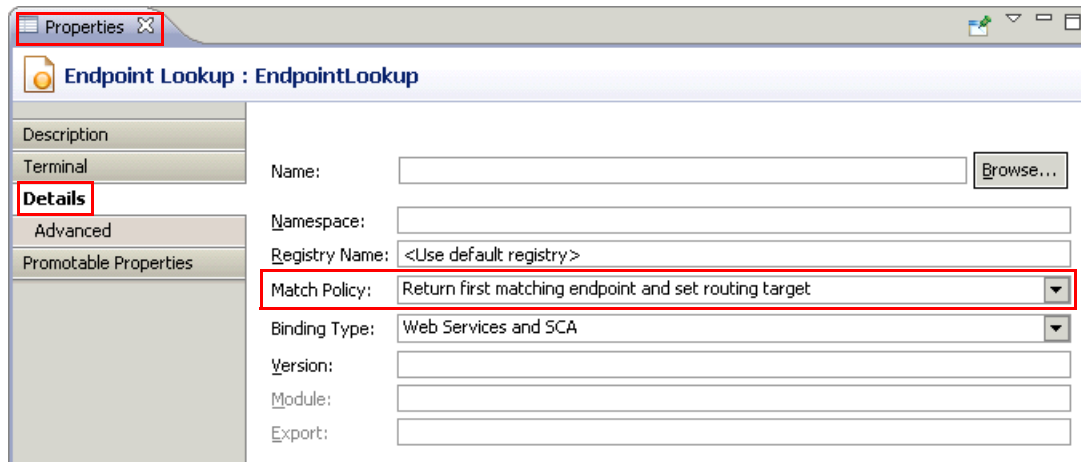


Figure 8-5 Configuration details for the Endpoint Lookup mediation primitive

If more than one service in WSRR matches the search criteria from the query, the Match Policy setting determines how many service endpoints are added to the outgoing result message. Table 8-1 summarizes the effect of the match policy on the outgoing result message.

Table 8-1 Match Policy types

Match Policy	Effect on the outgoing result message
Return all matching endpoints	<ul style="list-style-type: none"> ▶ Message context is updated with service information for all services returned by the registry ▶ Callout address in the message header is deleted ▶ Alternate targets list in the message header is deleted ▶ Outgoing result message must be processed to choose and set a service endpoint
Return first matching endpoint and set routing target	<ul style="list-style-type: none"> ▶ Callout address in the message header is updated with the first service address from the results returned ▶ Message context is updated with service information related to the address in the callout address ▶ Alternate targets list in the message header is deleted

Match Policy	Effect on the outgoing result message
Return all matching endpoints and set alternate routing targets	<ul style="list-style-type: none"> ▶ Callout address in the message header is updated with the first service address from the results returned ▶ Alternate targets list in the message header is updated with the remaining service addresses from the results returned ▶ Message context is updated with service information for all services returned by the registry
Return endpoint matching latest compatible version of SCA module-based services	<ul style="list-style-type: none"> ▶ Callout address in the message header is updated with the service address that has the highest version from the results returned ▶ Message context is updated with service information related to the address in the callout address ▶ Alternate targets list in the message header is deleted

If no service in WSRR matches the query, the output terminal (noMatch) propagates the original incoming message. The Match Policy settings are not considered in this situation.

8.3.2 SLA Check mediation primitive

The SLA Check mediation primitive determines whether a service consumer has an appropriate SLA in place to access a requested service provider. A capability version, an SLA, and a service level definition (SLD) need to be defined in WSRR to enable the registry to be queried. The SLA Check mediation primitive queries WSRR for this information.

As shown in Figure 8-6, the SLA Check mediation primitive has one input terminal, one fail terminal (rectangle shaped terminal), and two output terminals (accept and reject). The input terminal is wired to accept a message, and the other terminals are wired to propagate a message. If the information passed on the incoming message is used to successfully find a matching SLA in WSRR, the original incoming message is propagated by the accept terminal. However, if no matching SLA is found, the reject terminal is fired and propagates the original incoming message.



Figure 8-6 SLA Check mediation primitive

To find a matching SLA in WSRR, the SLA Check mediation primitive uses information in the incoming message. Figure 8-7 shows the Details tab in the Properties view where the parameters for the search can be configured.

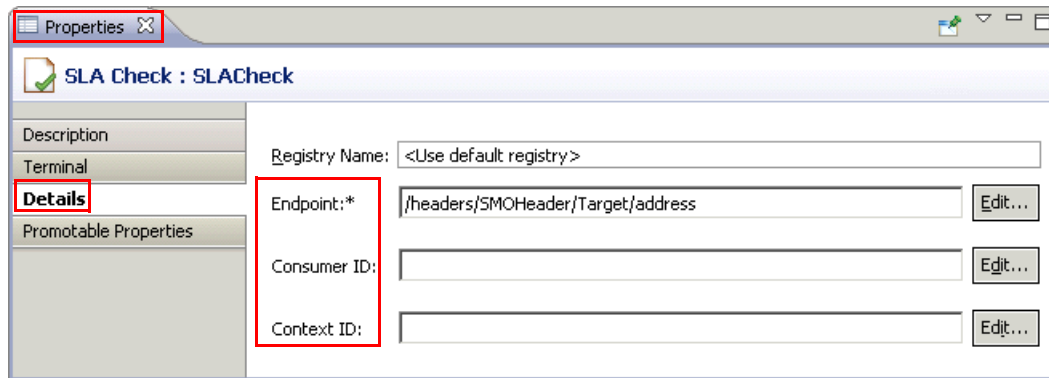


Figure 8-7 Configuration details for SLA Check mediation primitive

The SLA is matched on the following parameters:

Endpoint	This parameter can be a literal value or can be passed as part of the incoming message. This field is mandatory. If not set, the reject terminal is fired and the original incoming message is propagated.
Consumer Identifier	This parameter can be a literal value or can be passed as part of the incoming message. It identifies the service consumer of the target endpoint.
Context Identifier	This parameter can be a literal value or can be passed as part of the incoming message. It identifies the context under which the service consumer's invocation of the target endpoint occurs.

Setting the SLA Check endpoint: The SLA Check mediation primitive needs an endpoint already set in the incoming message to check against WSRR. The endpoint can be set in two ways:

- ▶ Set the endpoint statically to a specific value.
- ▶ Retrieve the endpoint dynamically from WSRR with the Endpoint Lookup mediation primitive.

However, for a dynamic endpoint selection at run time, the Endpoint Lookup mediation primitive must be implemented *before* the SLA Check mediation primitive within the mediation flow.

Setting the optional fields Consumer ID and Context ID is recommended to have a meaningful response. By not setting those fields the SLA Check mediation primitive will fire the accept terminal if it finds any existing SLA for the service provider requested, regardless of the requesting consumer. The result is that any consumer can call a service provider that has any active SLA without checking if a contract exists to allow the request.

8.3.3 Service Invoke mediation primitive and Callout node

The Service Invoke primitive and the Callout node are not specifically primitives that are used to increase agility between WSRR and the WebSphere ESB mediation module. However, they provide basic configuration options for endpoint selection that are retrieved from the WSRR.

Service Invoke mediation primitive

As shown in Figure 8-8, the Service Invoke mediation primitive has one input terminal and multiple output terminals. There is a fail terminal (rectangle-shaped terminal) for unmodeled faults and one output terminal for each modeled fault that has to be added manually. In addition, an output terminal is used for successful service calls.



Figure 8-8 Service Invoke mediation primitive

Consider using one or multiple Service Invoke mediation primitives if interaction with multiple services is necessary and output that combines service responses needs to be produced. In this type of configuration, a Callout node might not be required. When using the Service Invoke mediation primitive inside a mediation

flow, the input message is used to call the service. If the call is successful, the response is used to create the output message. If the call is unsuccessful, you can either retry the same service endpoint or call alternate service endpoints.

Callout node

As shown in Figure 8-9, the Callout node has only one input terminal and no output or fail terminals. In contrast to the Service Invoke mediation primitive, it switches from request flow to response flow. Therefore, the output of a Callout node is processed by the corresponding response flow.

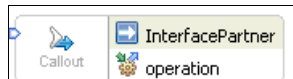


Figure 8-9 Callout node

The use of a Callout node rather than a Service Invoke mediation primitive is appropriate if you need to mediate a message and then call a service provider. The Callout node provides the most basic model for this configuration. When using a Callout node inside a mediation request flow, the input message is used to call the service. If the call is successful, the response is used to create the input message for the mediation response flow. If the call is unsuccessful, you can either retry the same service endpoint or call alternate service endpoints.

Dynamic endpoint options

The Service Invoke mediation primitive and the Callout node provide similar basic functionality for non-static endpoint selection. Figure 8-10 shows the option that can be configured for a successful service call.

Reference name:	InterfacePartner
Operation name:	operation
<input checked="" type="checkbox"/> Use dynamic endpoint if set in the message header	
Async timeout (seconds):	5
<input type="checkbox"/> Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.	
Invocation style:	Default

Figure 8-10 Endpoint selection option for a successful service call

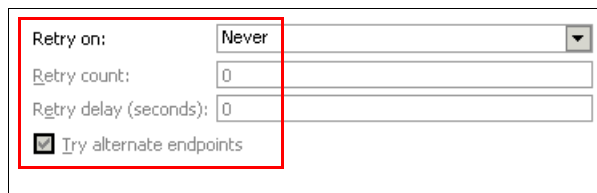
On the Details tab in the Properties view for either the Service Invoke mediation primitive or the Callout node you can find the option to use a dynamic endpoint if it set in the message header. By default, this option is enabled (checked). If there is no endpoint available in the message header, the static endpoint in the reference binding is chosen for the service call.

SMOHeader Note: The SMOHeader must be populated with the target address (/headers/SMOHeader/Target/address) and optionally the alternate target addresses (/headers/SMOHeader/AlternateTarget) prior to the callout or service invoke to make use of the “Use dynamic endpoint if set in the message header” property. Otherwise, the static default endpoint is used if it is provided by the wired import of the default service selected.

If a service call is unsuccessful, you can configure a retry on the Retry tab in the Properties view for either the Service Invoke mediation primitive or the Callout node. As shown in Figure 8-11, there are various parameters to set for a retry. To have a retry occur, you can set the following actions:

- ▶ Never
- ▶ Any fault
- ▶ Modeled fault
- ▶ Unmodeled fault

If Retry on is set to Never, it is not possible to select any alternate endpoint. In all three other cases, the “Retry count,” the “Retry delay,” and the “Try alternate endpoints” options become available. If the retry count is larger than zero, a retry is performed. The status of the “Try alternate endpoints” option determines whether the list of alternate endpoints that are stored in the SMOHeader is used for the retry.



Retry on:	Never	▼
Retry count:	0	
Retry delay (seconds):	0	
<input checked="" type="checkbox"/> Try alternate endpoints		

Figure 8-11 Endpoint selection option for an unsuccessful service call with retry

8.4 Implementing a basic mediation proxy module

In this section, we implement a mediation module with basic capabilities to retrieve endpoints dynamically and to check whether an SLA is existing and active.

We create the mediation library module that stores all the necessary artifacts from WSRM, such as interface descriptions, WSDLPortType definitions, and business objects. After populating the mediation library, we then create the mediation module using this library for its implementation. When the basic

implementation is complete, we deploy that module to the test environment that was created in 4.5.1, “Installing and configuring IBM Integration Designer” on page 102. Finally, we run a test against the WSRR and analyze the results.

Before you continue: The service registry instance contains the initial required information, such as WSDL documents and service metadata, as described in Chapter 7, “Registering services” on page 195.

For your convenience, we provide a WSRR export file (WSRRExport001.zip) that contains all the required definitions. You can find it in the /Basic_Mediation/StartingPoint folder of the Additional material file.

To import this file into WSRR, launch the WSRR console and in the Administrator perspective, select **Actions** → **Import** and import WSRRExport001.zip.

8.4.1 Creating the mediation library module

This section describes how to create the mediation library module and populate it with the artifacts from WSRR. The library is then used as a dependency for the mediation proxy module, providing service interface definitions, business objects, and web service port definitions.

Importing artifacts from the registry using the WSRR Eclipse Plug-in

The WSRR Eclipse Plug-in is used to connect to a WSRR instance at development time and to query that registry to retrieve service information and physical service artifacts. It can also be used to manage relationships and to modify service artifacts. These changes can be updated to the corresponding WSRR instance.

Before you continue: The test environment runtime server that was created in 4.5.1, “Installing and configuring IBM Integration Designer” on page 102 must be started successfully at development time to retrieve artifacts from WSRR into the workspace of Integration Designer.

To import artifacts from the registry using the WSRR Eclipse Plug-in, follow these steps:

1. Start Integration Designer, and switch to the Business Integration perspective.
2. Click **Window** → **Show View** → **Other** as shown in Figure 8-12.

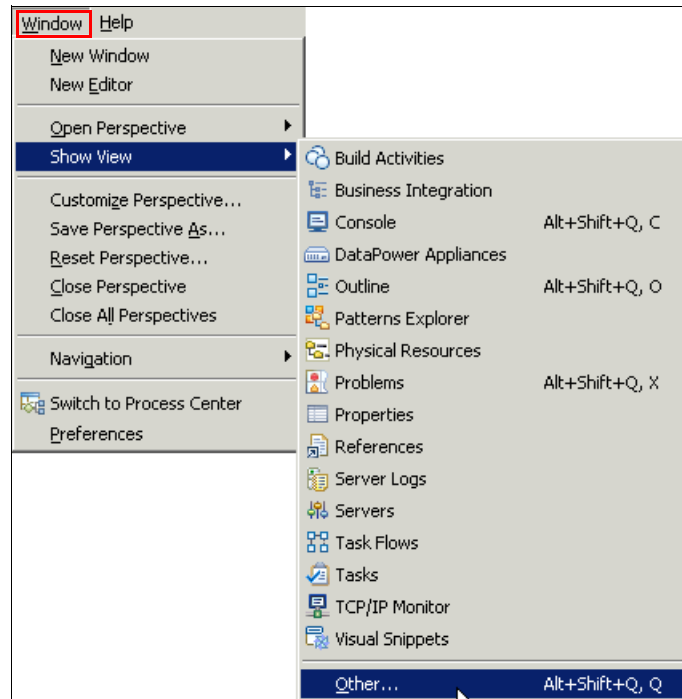


Figure 8-12 Show view Other

3. Expand **WebSphere Service Registry and Repository (WSRR)**, select **WSRR Content**, and click **OK**, as shown in Figure 8-13.

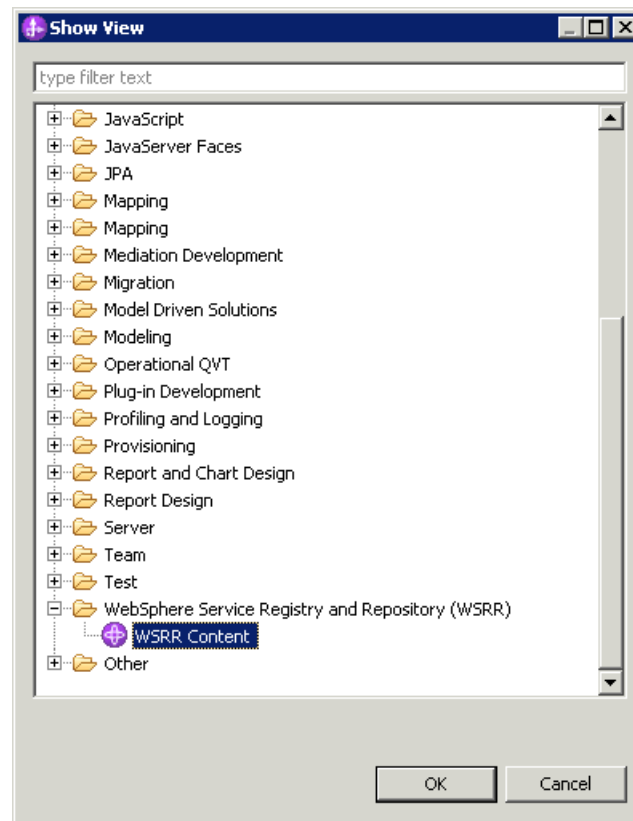


Figure 8-13 Show view WSRR Content

4. Switch to the WSRR Content view and right-click in the panel. Be sure to connect to the correct WSRR definition by pointing to **Choose WSRR**. Select **Retrieve** from the options, as shown in Figure 8-14.

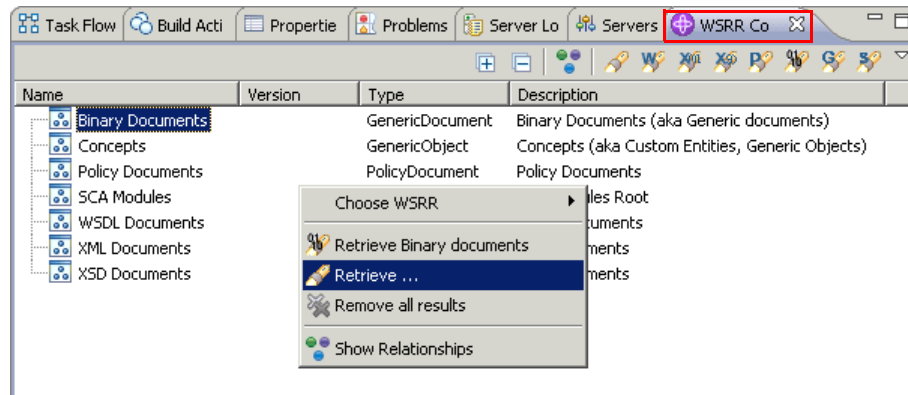


Figure 8-14 WSRR search for artifacts

5. After selecting the Retrieve option, a search window opens. You can use the search criteria to query a WSRR instance to search for specific services. To retrieve all service information and service artifacts stored in the WSRR instance, select all options under the Search for section, as shown in Figure 8-15.

Click **Search** to start the query.

Search query note: You can specify the search query using various parameters. If you have a WSRR instance with many services, you can specify search parameters, such as the name of the service or the version of the service, and you can narrow the choice of documents that you might want to retrieve. For our discussion in this book, searching for all services in the registry is sufficient, because there is only one service.

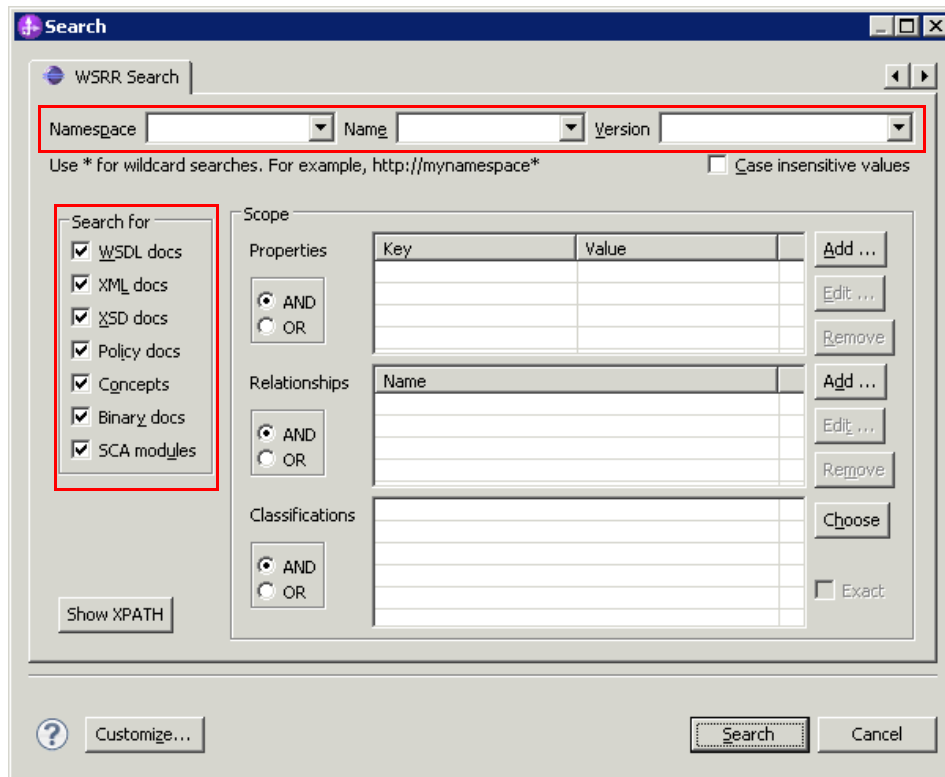


Figure 8-15 Search criteria for lookup in WSRR

The search retrieves all documents and artifacts that are available in WSRR. Browse through the various document types that were received. Figure 8-16 shows the necessary documents for the module library, such as WSDL documents and XSD documents.

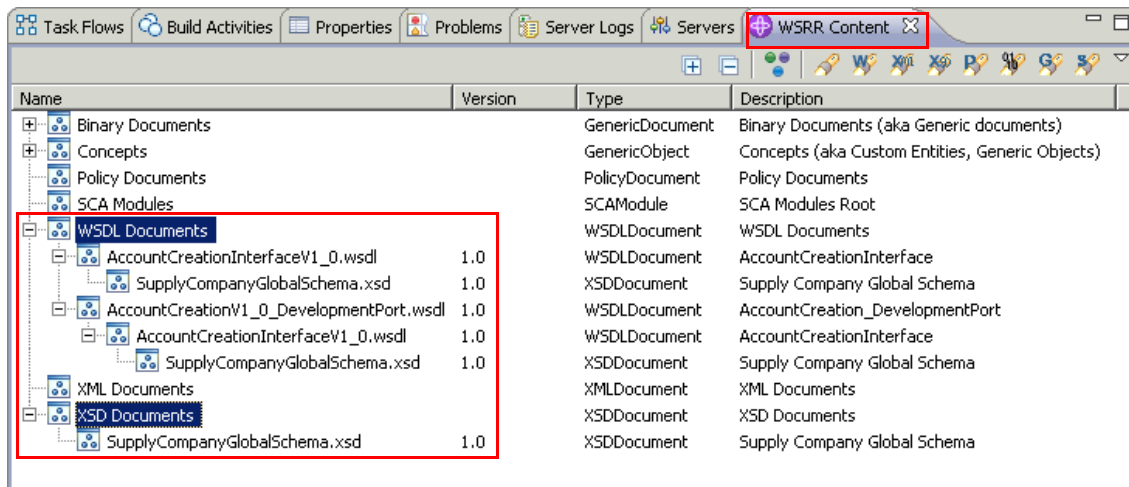


Figure 8-16 Results retrieved from WSRR

Creating the mediation library module

The mediation library module contains artifacts such as web service definitions, business objects, and interface descriptions, that are necessary to build the mediation proxy module for the Account Creation Service.

To create the mediation library module:

1. Right-click the blank Business Integration tab, and then select **New** → **Library**, as shown in Figure 8-17.

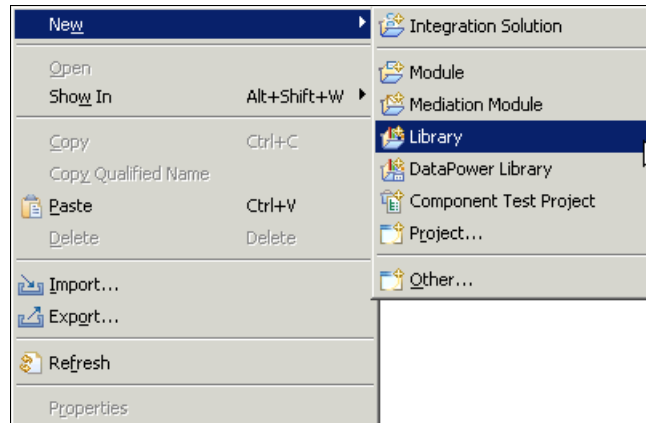


Figure 8-17 Creating the mediation library module

2. Set the library name to AccountCreationService_Med_Lib, and click **Next** (Figure 8-18).

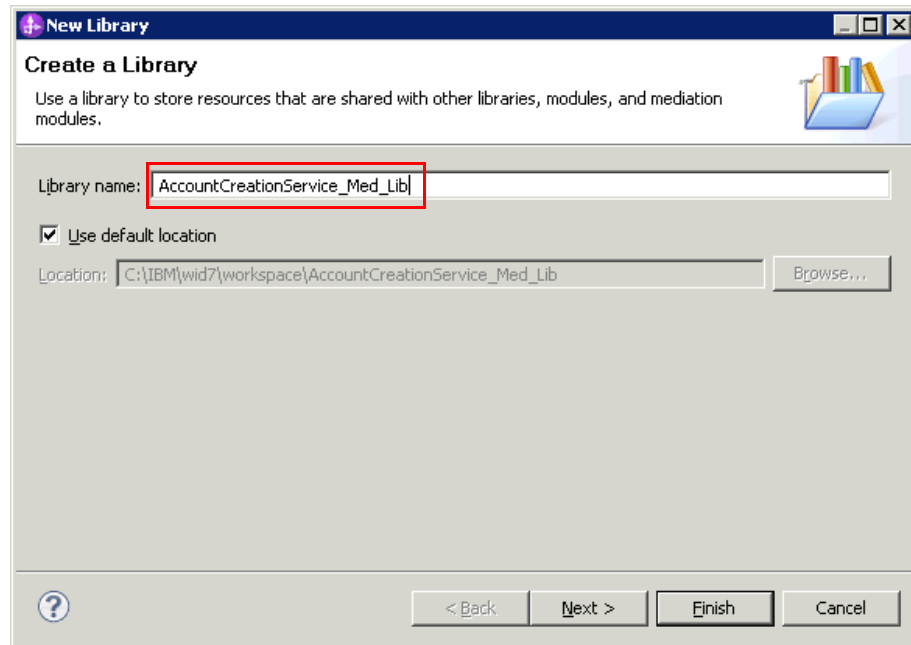


Figure 8-18 Specifying the library modules name and default location

3. Choose **Eager parsing** as the preferred business object parsing mode and deselect **Lazy parsing** as shown in Figure 8-19, and click **Finish**.

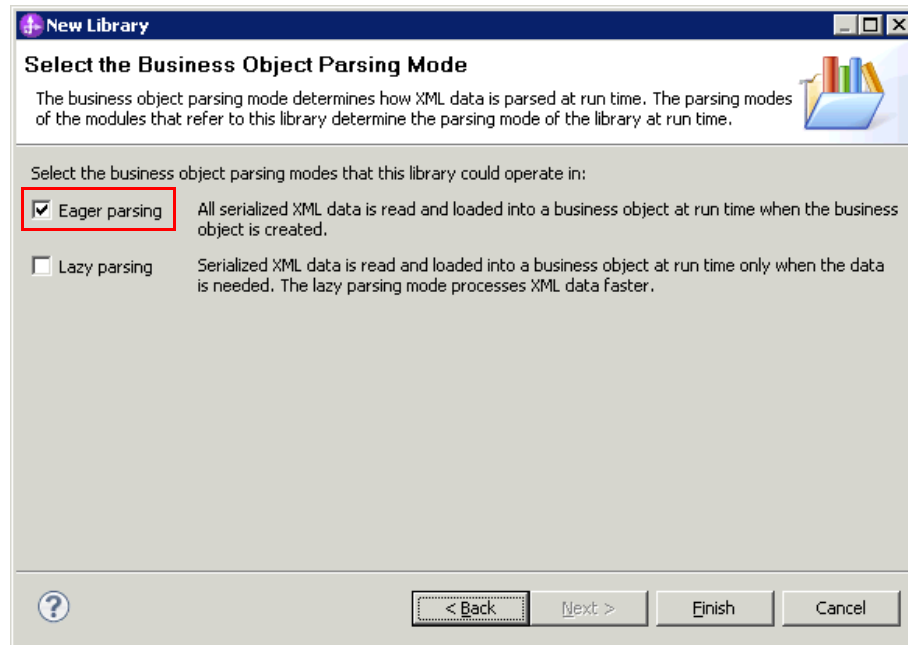


Figure 8-19 Select the business object parsing mode

A mediation library module called *AccountCreationService_Med_Lib* displays in the Business Integration tab as shown in Figure 8-20.

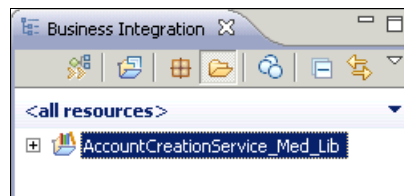


Figure 8-20 Business Integration tab containing the library module

4. The WSRR Eclipse Plug-in allows the import of successfully retrieved service artifacts in a mediation library module. To import the necessary files in the library module, switch to the WSRR Content view. Right-click **AccountCreationServiceV1_0_DevelopmentPort.wsdl**, and select **Import Document** as shown in Figure 8-21.

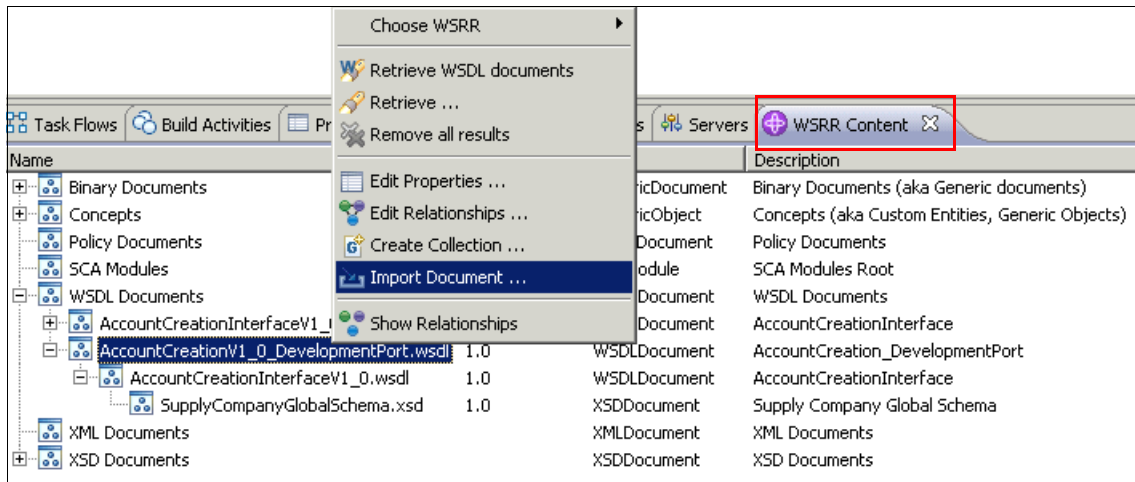


Figure 8-21 Import documents from WSRR Eclipse Plug-in

- Next, select the AccountCreationService_Med_Lib module. Select the **Include all dependent artefacts/entities** and **Generate folder structure** options, as shown in Figure 8-22. Click **Finish**.

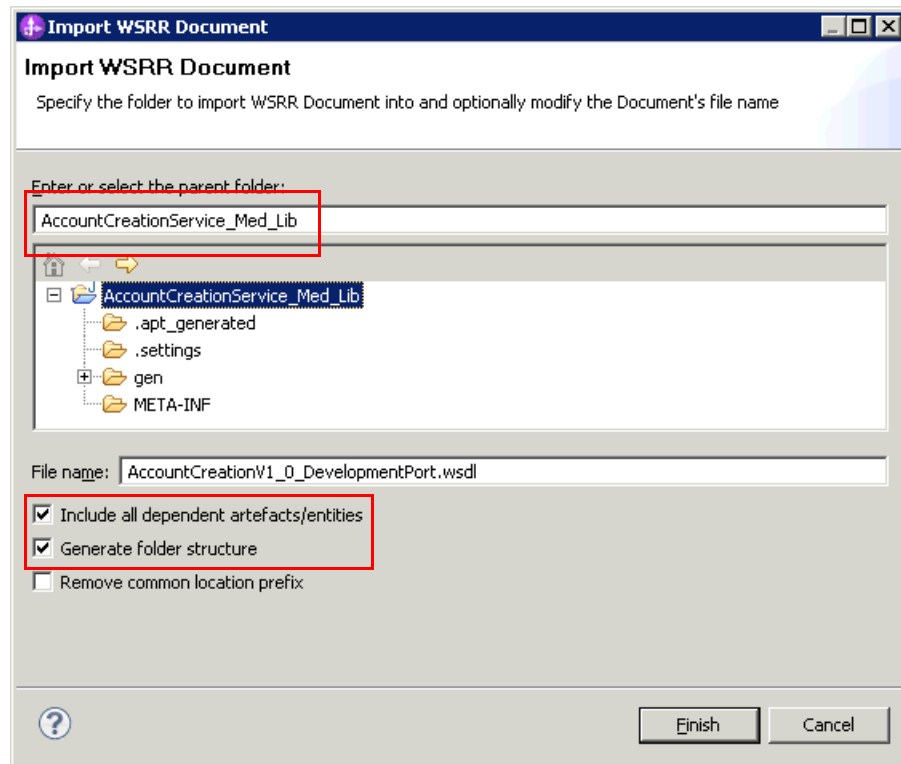


Figure 8-22 Import WSRR document

After the necessary artifacts are imported, review the mediation library module in the Business Integration tab. The library now contains several business objects, an interface, and a web service port definition as shown in Figure 8-23.

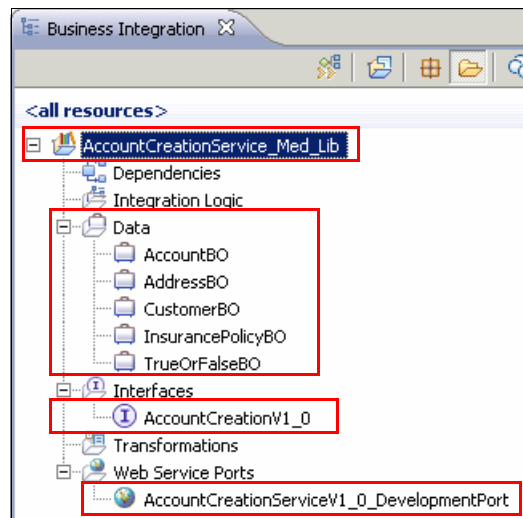


Figure 8-23 Review content for the mediation library module

Creating soapHeader objects

The SLA Check primitive can check whether a specific SLA exists between a service consumer and a service provider. Furthermore, it can check whether a specific SLA exists in a certain context. Therefore, a client needs to provide a consumer ID and a context ID. This data is stored in the SOAP header of the incoming message to the mediation proxy module.

To create a business object that contains these two parameters:

1. Go to the Business Integration tab, and right-click **Data**.
2. Select **New** → **Business Object** as shown in Figure 8-24.

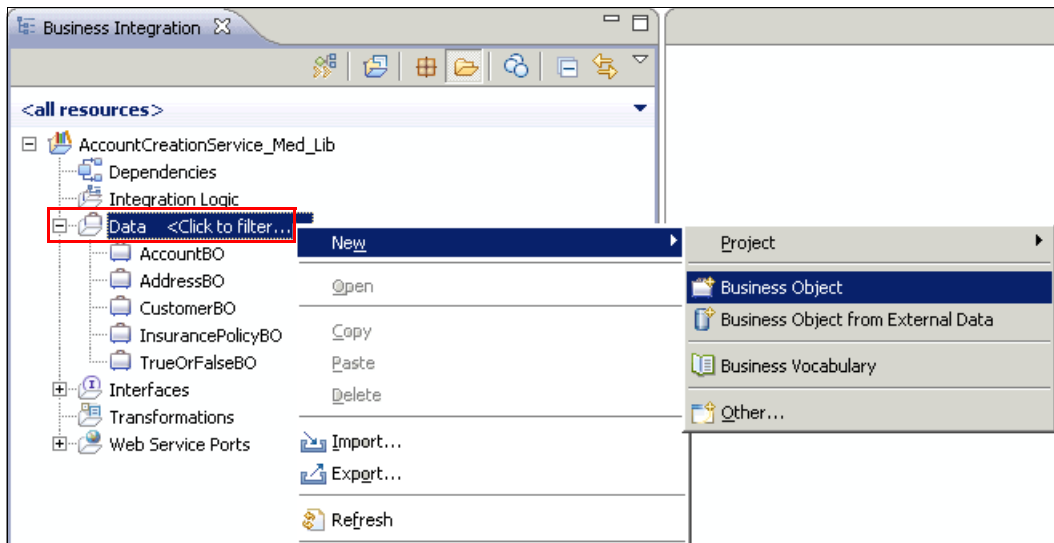


Figure 8-24 Create new business object for soapHeader

3. Set parameters, such as name of the business object, namespace, or as needed. We entered the following parameters (Figure 8-25):

- Namespace: `http://SupplyCompany.itso.ibm.com/Account`
- Folder: `soapHeader`
- Name: `GEPHeaderBO`

Click **Finish**.

New Business Object

Create a Business Object

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library:

Namespace: ☐ Default

Folder:

Name:

Inherit from:

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. [More...](#)

☐ Create the business object as an element

Figure 8-25 Create a business object


4. The newly created and empty business object needs to be populated with two attributes. By clicking the  icon in the Business Object editor, you can add attributes to the existing business object. Create the following attributes of type string as shown in Figure 8-26.
 - Create an attribute of type string and set consumerID as the field name.
 - Create an attribute of type string and set contextID as the field name.



Figure 8-26 GEPHeaderBO containing two attributes of type string

5. Save the changes in the Business Object editor.

8.4.2 Creating the basic mediation proxy module

This section describes how to create the mediation proxy module. First, we create the module skeleton and set the dependency to the mediation library module. After adding a web service export component and the corresponding web service binding, we implement the mediation flow component. This process involves integrating dynamic endpoint lookup and SLA checks. Finally, we add basic tracing and error handling functions for debugging purposes only.

Creating the mediation module

To create the mediation module for the Account Creation Service:

1. Switch to the Business Integration tab, and right-click any blank space. Then, choose **New** → **Mediation Module** from the context menu, as shown in Figure 8-27.

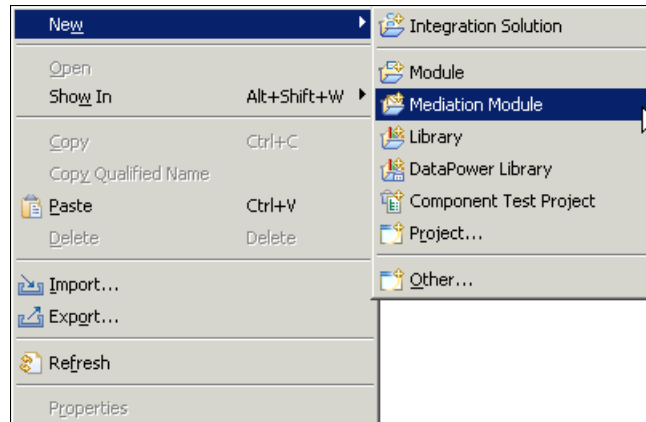


Figure 8-27 Creating the mediation module

2. Next, specify parameters such as the module name, the target runtime environment, or the mediation flow component name, as shown in Figure 8-28:
 - a. Set the Module name to AccountCreationService_Med, and set the Name for the mediation flow component to ACS_Proxy.
 - b. Select **WebSphere ESB Server 7.5** from the drop-down list as the target run time.
 - c. Click **Next**.

New Mediation Module

Create a Mediation Module

Use a mediation module to integrate and connect services. A mediation module can contain mediation flows and can be deployed on WebSphere ESB or IBM Process Server.

Module name: AccountCreationService_Med

☒ Use default location

Location: C:\IBM\wid7\workspace\AccountCreationService_Med Browse...

Target runtime environment: WebSphere ESB Server v7.5

☒ Create mediation component

Name: ACS_Proxy

☒ Open the module assembly diagram

? < Back Next > Finish Cancel

Figure 8-28 Specifying the mediation modules parameters

To use the artifacts that are retrieved from WSRR and stored in the AccountCreationService_Med_Lib mediation library module, you need to add this library as a dependency to the newly created AccountCreationService_Med mediation module.

3. Select **AccountCreationService_Med_Lib** from the list of libraries, and click **Finish** (see Figure 8-29).

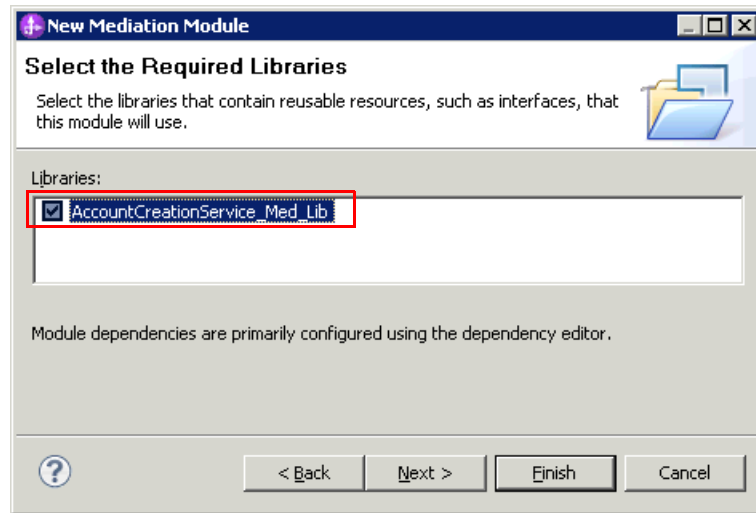


Figure 8-29 Adding a dependency

You have now created a mediation module with a dependency to the mediation library module that was created in 8.4.1, "Creating the mediation library module" on page 265.

Review the Business Integration tab for the mediation module. The Assembly Diagram for the module AccountCreationService_Med is opened, as shown in Figure 8-30.

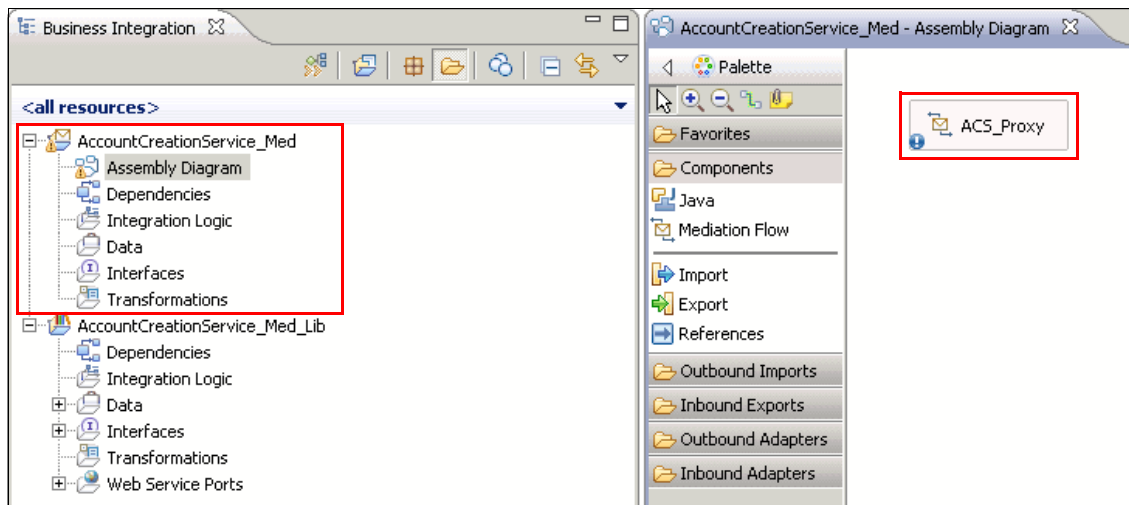



Figure 8-30 Business Integration tab and mediation module assembly diagram

Assembling web service exports and web service imports

By adding a service interface and a corresponding web service export to the mediation flow component, the module exposes its services to any client that needs to call the mediation proxy module. Adding a service reference to the mediation flow component allows the mediation module to invoke a specific reference partner for a specific service.

To add a service interface to the mediation flow component:

1. Point to the component, and click the  icon. The “Add interface” window opens, offering any interface that is available either in the mediation module or the dependent mediation library module.

2. Select the **AccountCreationV1_0** interface from the list of matching interfaces, and click **OK** as shown in Figure 8-31.

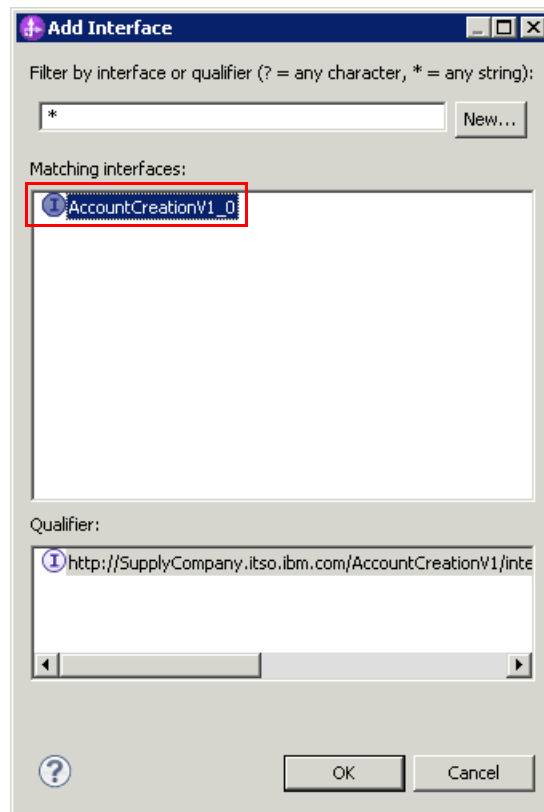



Figure 8-31 Choosing the corresponding interface

3. To add a service reference to the mediation flow component, point to the component, and click the  icon.

As with adding a service interface to the component, an “Add Reference” window opens to choose from the available reference partners.

4. Provide a reference Name (or use the default).

5. Select the **AccountCreationV1_0** interface from the matching interface list, as shown in Figure 8-32. Click **OK** to add the reference to the mediation flow component.

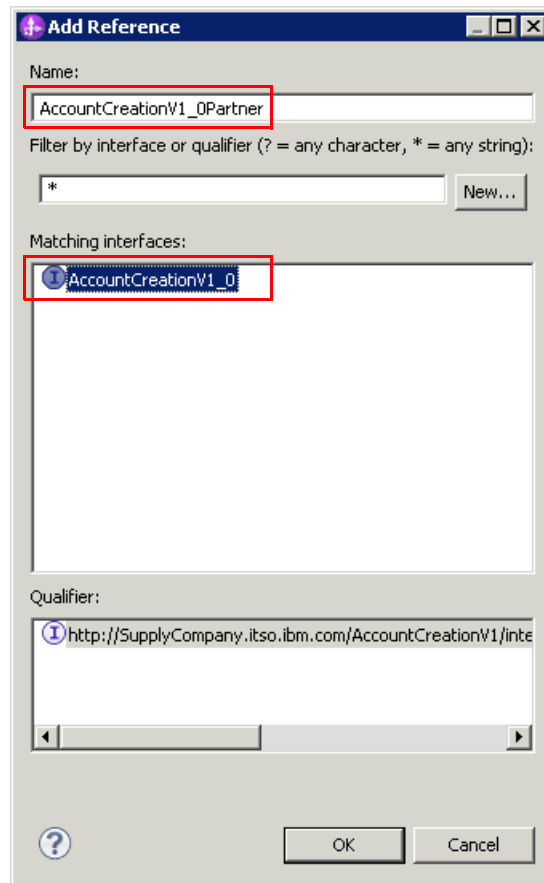


Figure 8-32 Choosing the corresponding reference partner

6. After successfully adding a service interface and a reference partner, you next need to expose the service interface through a web service export. To add an export component, go to the Palette in the Assembly Diagram tab. Open **Components**, and drag the Export onto the canvas.
7. Rename the Export to ACS_WS_EXP.
8. Wire the Export to the service interface on the ACS_Proxy mediation flow component.

Note: The Export does not have a specific service interface bound to it yet. If you add a wire between the export component and the mediation flow component, you are prompted to allow the target services to be used in other modules. Click **OK** in the message window to add the same service interface from the mediation flow component to the Export component.

You can also specify the service interface *before* wiring the export component to the mediation flow component. In this case, you are not prompted with this message.

To add an import component, go to the Palette again and complete these steps:

1. Open Components and drag the Import onto the canvas.
2. Rename the Import to ACS_WS_IMP.
3. Wire the WSDL Reference on the ACS_Proxy mediation flow component to the Import. Click **OK** to add the same service reference interface from the mediation flow component to the Import component.

After completing the previous steps, the assembly diagram for the AccountCreationService_Med looks as shown in Figure 8-33.

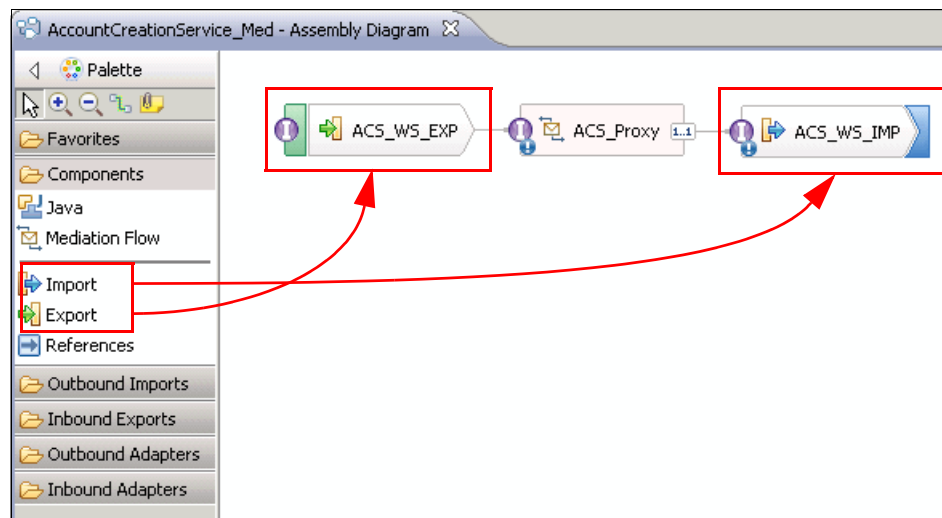


Figure 8-33 AccountCreationService_Med assembly diagram

To generate a web service binding for the export component:

1. Right-click the **ACS_WS_EXP** component, and select **Generate Binding** → **Web Service Binding** from the menu as shown in Figure 8-34.

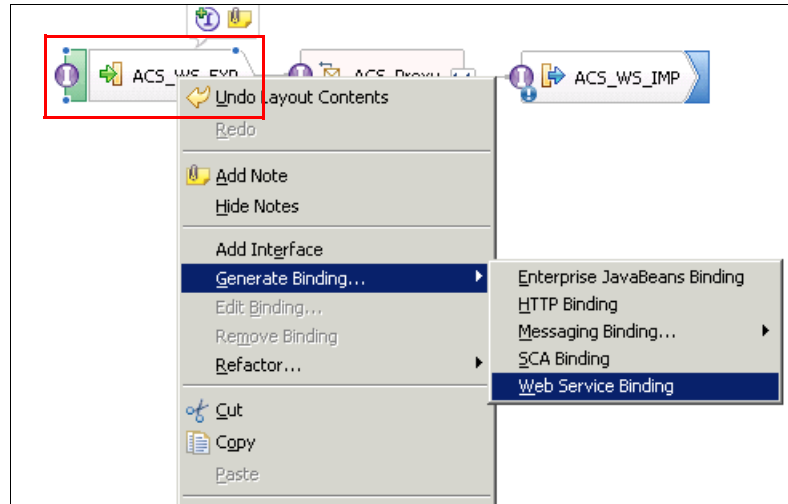


Figure 8-34 Generate web service binding for export component

2. You need to select a transport protocol that the ACS_WS_EXP uses for the web service binding. Select **SOAP1.1/HTTP** and click **Finish** to select the protocol, as shown in Figure 8-35.

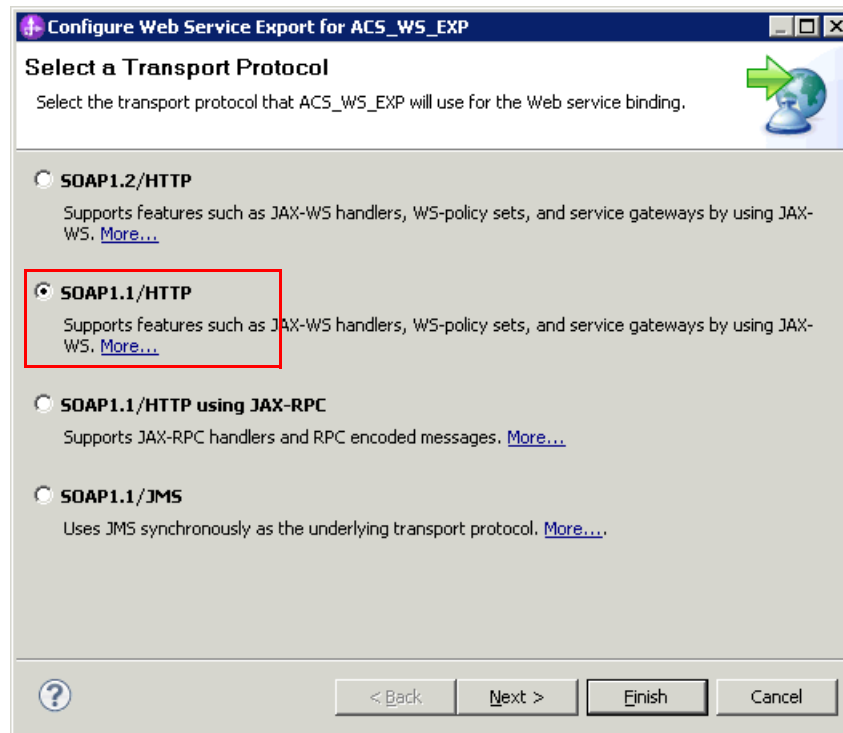


Figure 8-35 Web service export component configuration

Examine the Web Service Ports in the mediation library module AccountCreationService_Med_Lib in the Business Integration tab on the left. In addition to the existing port definitions that are retrieved by the WSRR Eclipse Plug-in, a new web service port definition for the ACS_WS_EXP export component is available (see Figure 8-36).

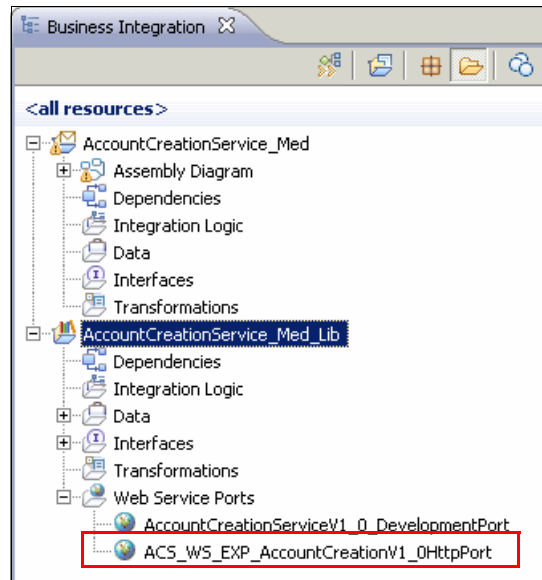


Figure 8-36 Web service port definition added to the mediation library module

This web service port is exposed to any potential client using the service offered by the AccountCreationV1_0 interface.

ACS_WS_IMP import component note: The mediation proxy module implements dynamic endpoint retrieval from WSRR. However, we do not generate a static web service binding for the import component, because the result of that generation is a static endpoint set in the web service binding. Therefore, the ACS_WS_IMP import component functions as a place holder only.

Generating skeleton implementation for mediation flow

To generate a implementation for the ACS_Proxy mediation flow component:

1. Double-click the ACS_Proxy component in the assembly diagram for AccountCreationService_Med.
2. When prompted, click **Yes** to create the skeleton implementation for the mediation flow component, as shown in Figure 8-37.

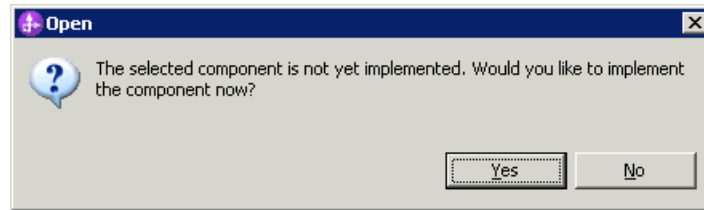


Figure 8-37 Implementation of the mediation flow component

3. Next, either provide a folder where the generated implementation is stored or leave it as default, and click **OK**.

Default location: By default the generated implementation is stored to the root folder of the AccountCreationService_Med mediation module.

4. The Overview tab in the skeleton implementation contains the service interface on the left and the reference partner on the right. To mediate between the two, click **createAccount** within the service interface AccountCreationV1_0. An information dialog box opens that contains three implementation templates, as shown in Figure 8-38.

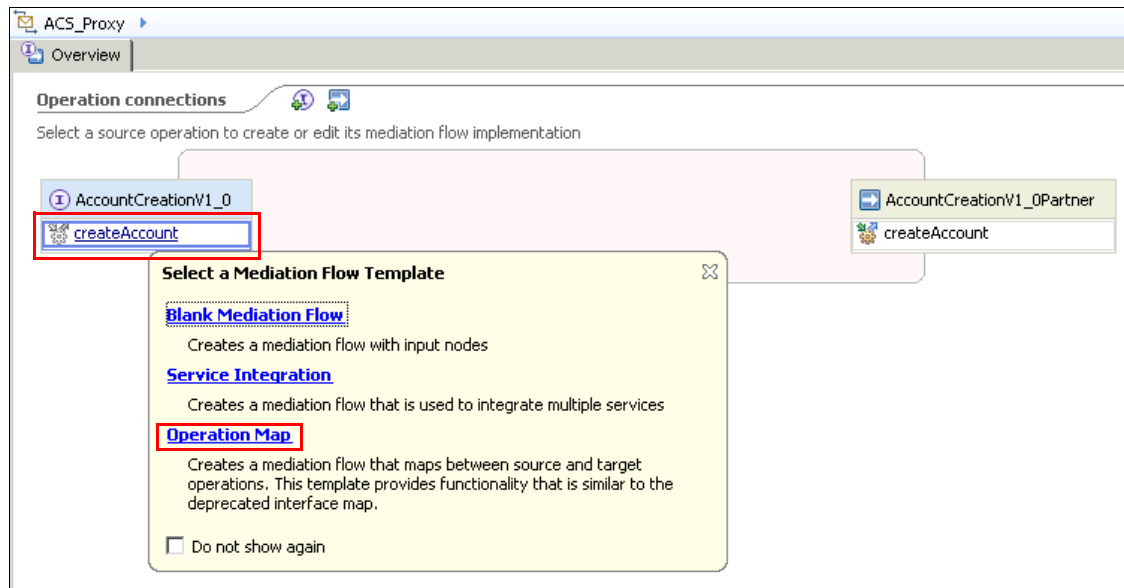


Figure 8-38 Operational map for implementation of ACS_Proxy

5. Select the **Operation Map** option to create a mediation flow that maps between the source and the target operation.

6. Choose the **createAccount** operation from the list of target operations, and click **OK** as shown in Figure 8-39.

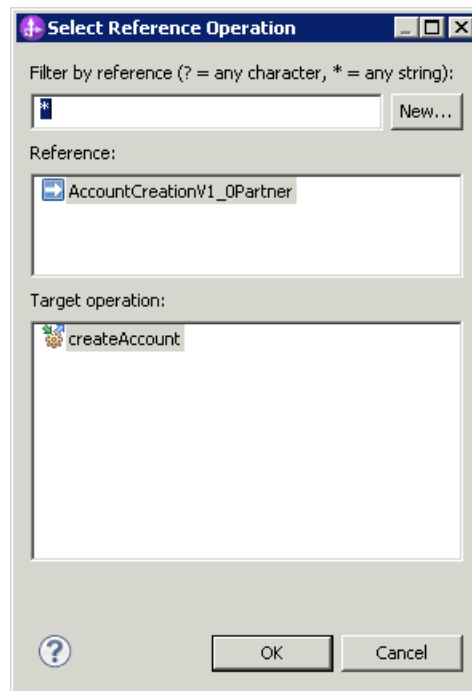


Figure 8-39 Select reference operation for operational mapping

After the Operation Map implementation is generated, an empty mediation flow, similar to that shown in Figure 8-40, is created.

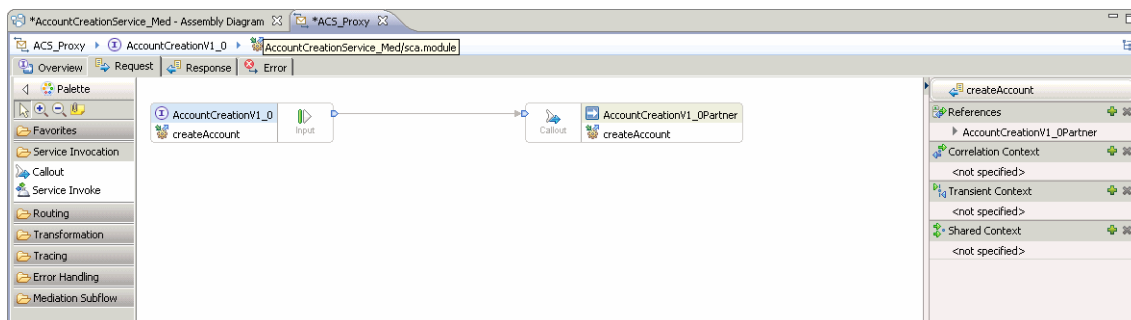


Figure 8-40 Blank implementation of mediation flow component

Integrating the Endpoint Lookup primitive

To add a dynamic endpoint lookup to the mediation flow implementation:

1. Go to the Palette in the ACS_Proxy implementation tab.
2. Open **Routing**, and drag the **Endpoint Lookup** primitive onto the canvas.
3. Rename the Endpoint Lookup primitive to EndpointLookupACS.
4. Wire the Endpoint Lookup primitive to the Input and to the Callout node in the mediation flow component as shown in Figure 8-41.

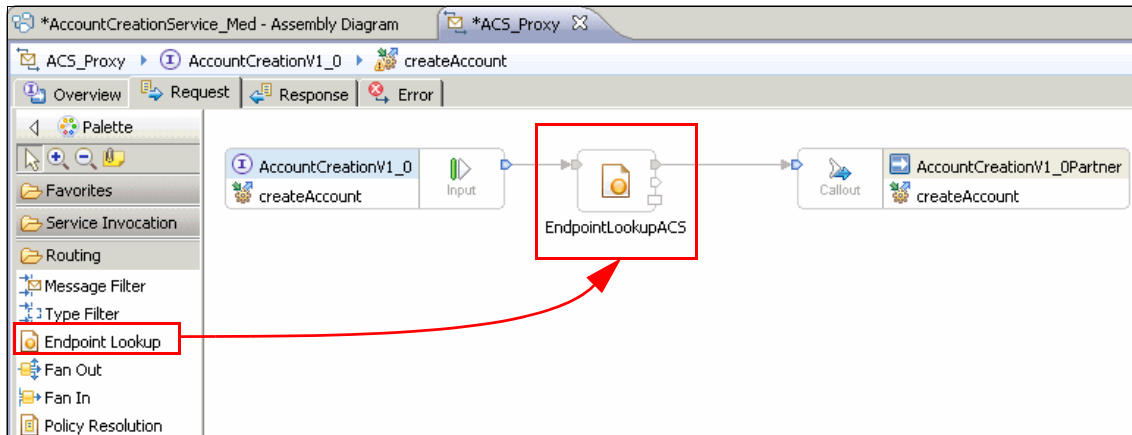


Figure 8-41 Adding the Endpoint Lookup primitive

The EndpointLookupACS mediation primitive looks for the AccountCreationV1_0 service in the WSRR instance. To configure the primitive:

1. Click the **EndpointLookupACS** mediation primitive, and go to the Properties view.
2. On the Details tab, click **Browse**.
3. Choose the **AccountCreationV1_0** interface to specify the name and namespace for the service the EndpointLookupACS mediation primitive that will query the WSRR instance.

4. Select **Return all matching endpoints and set alternate routing targets** as the Match Policy, as shown in Figure 8-42.

The screenshot shows the 'Endpoint Lookup' configuration window. The 'Properties' tab is active, and the 'Details' section is expanded. The configuration fields are as follows:

Field	Value
Name	AccountCreationV1_0
Namespace	http://SupplyCompany.itso.ibm.com/AccountCreationV1/interface
Registry Name	<Use default registry>
Match Policy	Return all matching endpoints and set alternate routing targets
Binding Type	Web Services and SCA
Version	
Module	
Export	

Figure 8-42 Configuring the Endpoint Lookup primitive

Integrating SLA Check primitive

To add an SLA check to the mediation flow implementation, go to the Palette in the ACS_Proxy implementation tab and complete these steps:

1. Open **Routing**, and drag the **SLA Check** mediation primitive onto the wire between the EndpointLookupACS mediation primitive and the Callout node.
2. Rename the SLA Check mediation primitive to SLACheckACS.

The SLACheckACS mediation primitive is now wired and named in the mediation flow as shown in Figure 8-43.

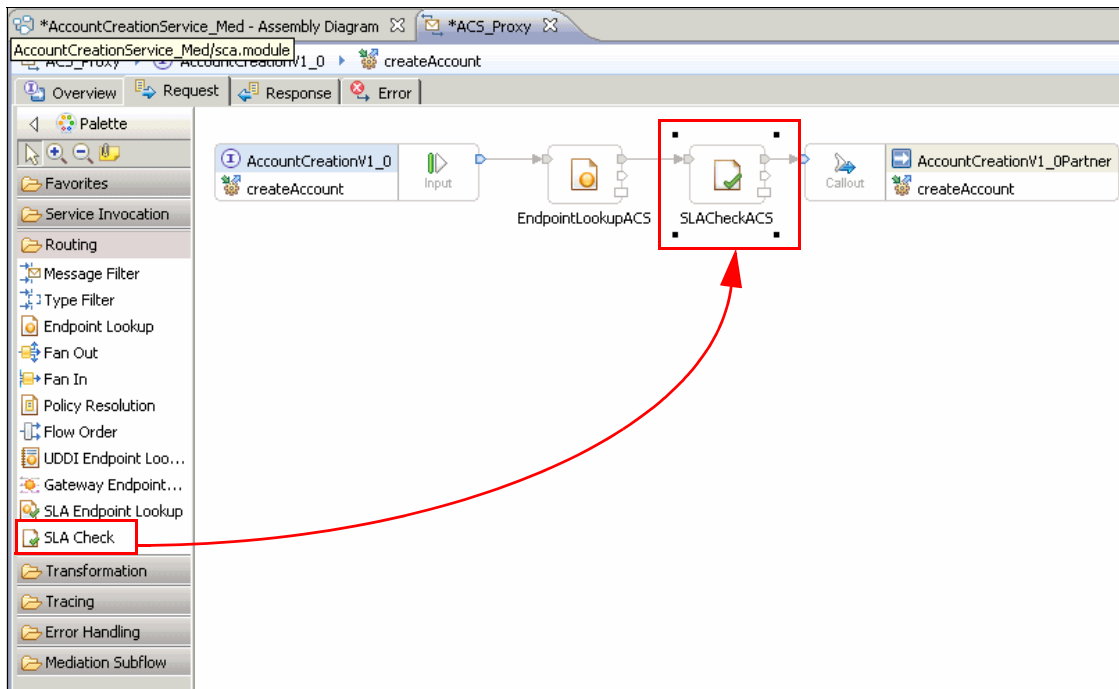


Figure 8-43 Adding the SLA Check primitive

The SLACheckACS mediation primitive looks for active AccountCreationV1_0 service SLAs in the WSRR instance. To configure the primitive:

1. Click the **SLACheckACS** mediation primitive, and go to the Properties view. Then, select the Details tab.
2. Set the following expression for the Consumer ID:
`/headers/SOAPHeader[name='GEPHeaderB0']/value/consumerID`
3. Set the following expression for the Context ID:
`/headers/SOAPHeader[name='GEPHeaderB0']/value/contextID`

The properties for the SLACheckACS mediation primitive are now configured as shown in Figure 8-44.

The screenshot shows the 'Properties' tab for the 'SLACheckACS' mediation primitive. The 'Details' section is selected, and the following fields are visible:

- Registry Name: <Use default registry>
- Endpoint:* /headers/SMOHeader/Target/address
- Consumer ID: /headers/SOAPHeader[name='GEPHeaderBO']/value/consumerID
- Context ID: /headers/SOAPHeader[name='GEPHeaderBO']/value/contextID

Figure 8-44 Configuring the SLA Check primitive

Specifying the path to Consumer ID and Context ID: The soapHeader uses the GEPHeaderBO created in the AccountCreationService_Med_Lib library module. To specify the path to Consumer ID and Context ID, you can either enter the values manually or use the Edit button and the XPath expression builder to assist you.

Configuring Callout node

To ensure that the Callout node uses the dynamically retrieved endpoints, follow these steps:

1. Click the **Callout** node, and go to the Properties tab. Then, select the Details tab.
2. Ensure that the **Use dynamic endpoint if set in the message header** option is selected as shown in Figure 8-45.

The screenshot shows the 'Properties' tab for the 'Callout : createAccount : AccountCreationV1_OPartner' node. The 'Details' section is selected, and the following fields are visible:

- Reference name: AccountCreationV1_OPartner
- Operation name: createAccount
- ☒ Use dynamic endpoint if set in the message header
- Async timeout (seconds): 5
- ☐ Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.
- Invocation Style: Default

Figure 8-45 Configuring the Callout node

- To use alternate endpoints retrieved and set in the message header, go to the Retry tab. Select the **Any fault** entry from the Retry on drop-down list. Set the retry count to the value of 3, and select the **Try alternate endpoints** options to perform a retry for the next three endpoints, as shown in Figure 8-46.

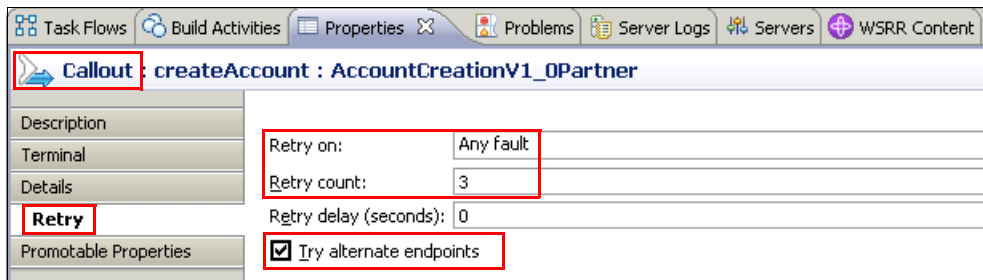


Figure 8-46 Configuring the Callout node's retry properties

- The AccountCreationService_Med mediation module has now implemented a basic dynamic endpoint lookup and an SLA validation for the first retrieved endpoint. Save changes in all open editors.

Adding Trace and Fail primitives

For the purpose of this book, we add basic logging and error handling capabilities to the mediation flow. We add tracing as a first step in the mediation flow to log the incoming message and to trace the last step in the mediation flow to log the outgoing message. The Fail primitive terminates the mediation flow in case of no matching endpoints, a rejection of the SLA check, or in case of failure of the EndpointLookupACS or SLACheckACS mediation primitive. Before the Fail primitive we add another trace primitive to log the failure message.

Figure 8-47 shows the complete ACS_Proxy mediation flow.

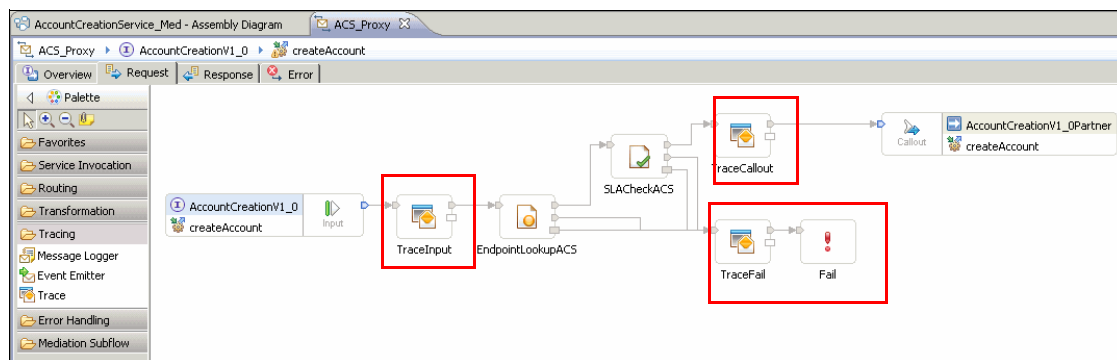


Figure 8-47 Adding trace primitives and a fail primitive

Implementing a logging solution: For the purpose of this book, we added only a basic tracing to see messages in the outline of the system console for debugging purposes. Implementing a proper logging solution does not contribute to the purpose of this mediation in terms of complexity and visibility of the mediation flow. However, implementing a proper logging solution is a suggested good practice for a productive mediation module.

This mediation module does not perform error handling and logging. However, preferred practice suggests to add a fault to the createAccount operation and to implement a proper fault handling scenario to handle faults such as noMatch in the EndpointLookupACS primitive or reject in the SLACheckACS primitive. In addition, proper fault handling is necessary to handle runtime failures while executing those primitives.

8.4.3 Testing the mediation proxy module

This section describes how to test the mediation module. First, we install the service provider into the runtime environment. Next, we deploy the mediation proxy module AccountCreationService_Med to the test environment server. After running the test, we discuss the results of each implemented mediation primitive.

Installing the Account Creation Service

The Account Creation Service is already implemented and needs to be installed as an enterprise archive file (.ear) to the environment. The mediation proxy module calls the service provider Account Creation Service based on a retrieved endpoint. In this step, we install the service provider to this endpoint.

To install the Account Creation Service, go to the administrative console for your server profile. Then follow these steps:

1. Log on to the Console with your authentication credentials, and go to **Applications** → **Application Types** → **WebSphere enterprise applications**. Examine the applications that are installed.
2. Click **Install** as shown in Figure 8-48 to install the service provider Account Creation Service.

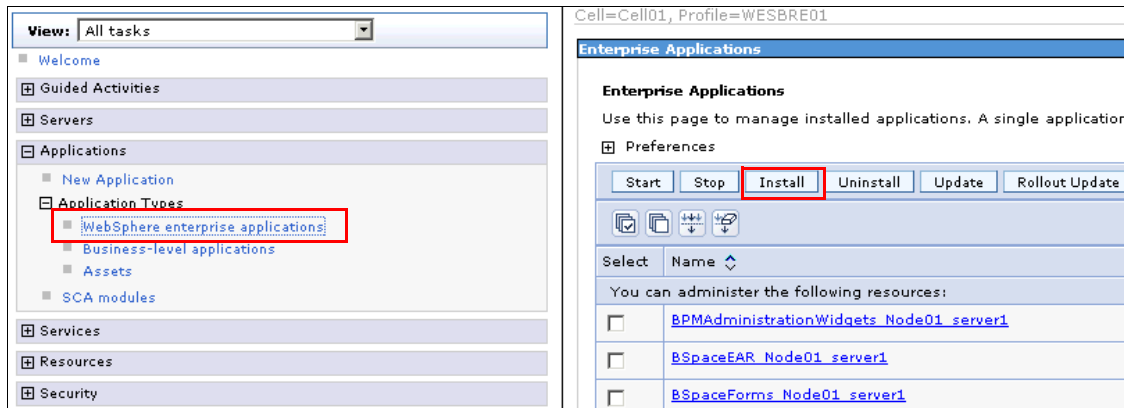


Figure 8-48 Applications on the administrative console

3. Click **Browse**, and navigate to the AccountCreationV1_OEAR.ear file on the file system as shown in Figure 8-49.

Downloading the file: The AccountCreationV1_OEAR.ear file is provided with this book. For information about how to download the resources that are available with this book, refer to Appendix A, “Additional material” on page 395.

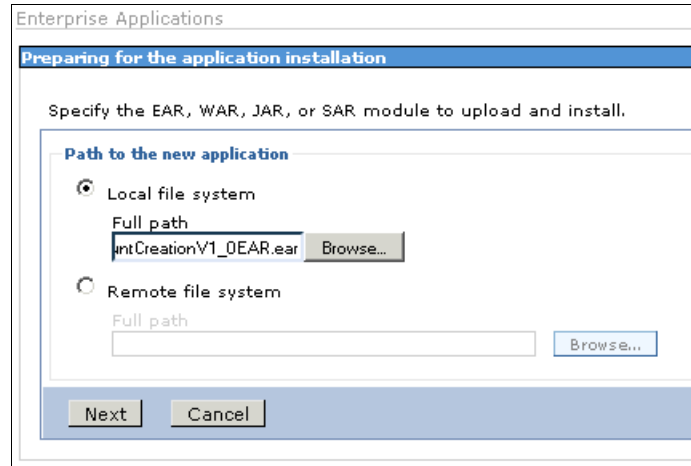


Figure 8-49 Location of the enterprise archive file for Account Creation Service

4. Click **Next** to continue the installation of the service provider.
5. Select the **Fast Path - Prompt only when additional information is required** option for the installation of the service provider, and click **Next**.

6. Click **Step 3 Summary**, and then click **Finish**, as shown in Figure 8-50.

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options
Step 2 Map modules to servers
→ Step 3: Summary

Summary

Summary of installation options

Options	Values
Precompile JavaServer Pages files	No
Directory to install application	
Distribute application	Yes
Use Binary Configuration	No
Deploy enterprise beans	No
Application name	AccountCreationV1_OEAR
Create MBeans for resources	Yes
Override class reloading settings for Web and EJB modules	No
Reload interval in seconds	
Deploy Web services	No
Validate Input off/warn/fail	warn
Process embedded configuration	No
File Permission	.*\,dll=755#.*\,so=755#.*\,a=755#.*\,sl=755
Application Build ID	Unknown
Allow dispatching includes to remote resources	No
Allow servicing includes from remote resources	No
Business level application name	
Asynchronous Request Dispatch Type	Disabled
Allow EJB reference targets to resolve automatically	No
Cell/Node/Server	Click here

Previous Finish Cancel

Figure 8-50 Finish the installation of the service provider

7. After the successful installation of the AccountCreationV1_OEAR.ear file, click **Save** to save the changes directly to the master configuration.

8. In the Enterprise Applications view, review the Application Status of the AccountCreationV1_OEAR application. Notice that the application has not yet started. To start the application, select the **AccountCreationV1_OEAR** application, and click **Start**, as shown in Figure 8-51.

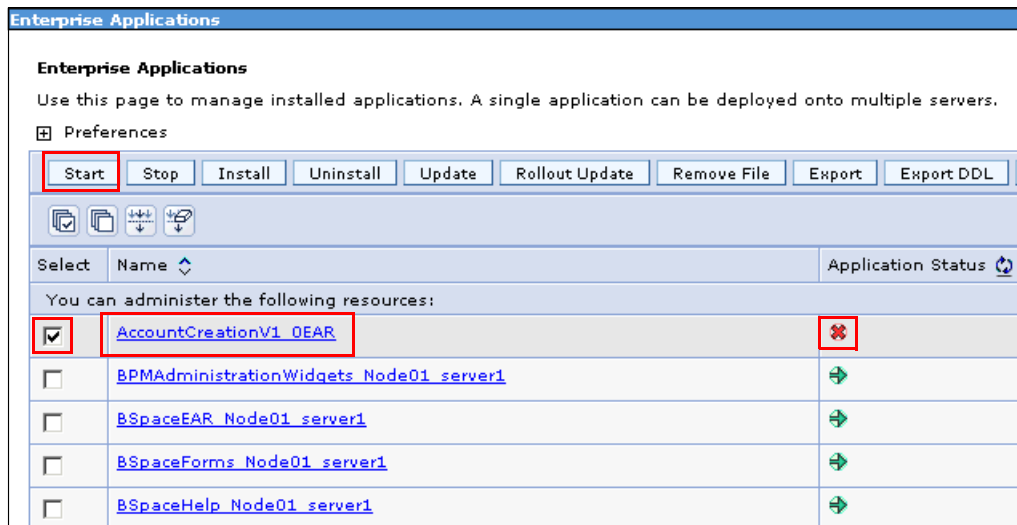


Figure 8-51 Starting the application AccountCreationV1_OEAR

After successfully starting the AccountCreationV1_OEAR application, the message shown in Figure 8-52 displays.

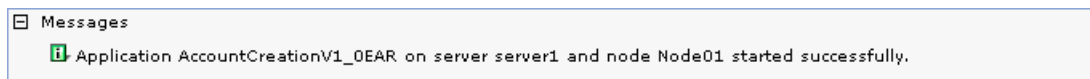


Figure 8-52 Application started successfully

The service provider is now running and can be called by the mediation proxy.

Deploying the AccountCreationService_Med

To test the implementation of the mediation proxy module AccountCreationService_Med, we need to deploy the mediation to the server. To add the module to the server:

1. Go to the Servers view. Right-click **WebSphere ESB v7.5 Server at localhost**, and select **Add and Remove** from the context menu.
2. Select the **AccountCreationService_MedApp** entry from the list of Available resources, and click **Add** to add the application to the server configuration, as shown in Figure 8-53.
3. Click **Finish**.

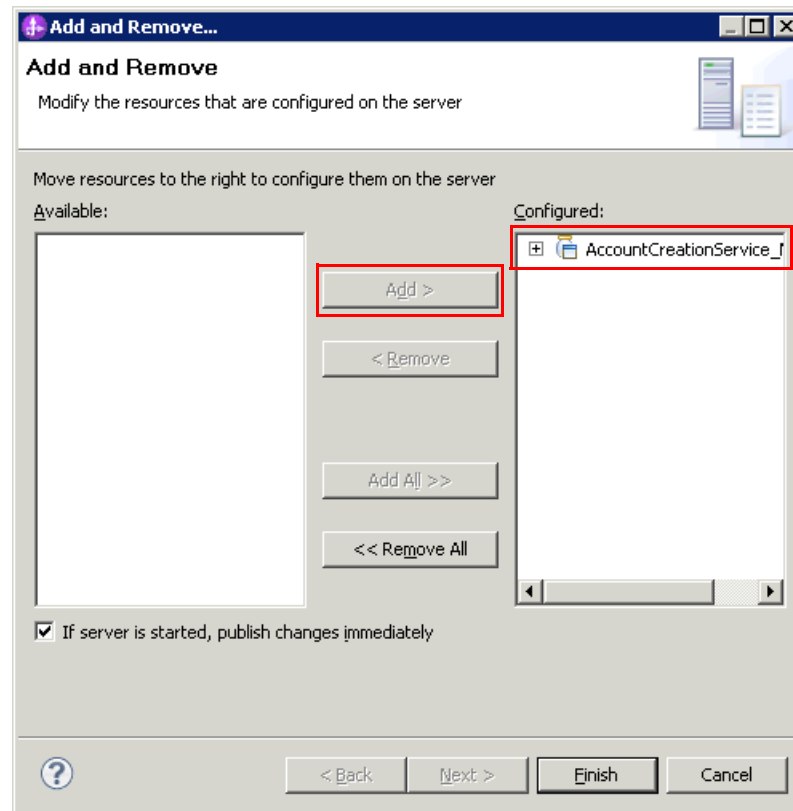


Figure 8-53 Add the application AccountCreationService_Med to the server

4. Review the application status in the Servers view. Wait until it is Started and Synchronized as shown in Figure 8-54.

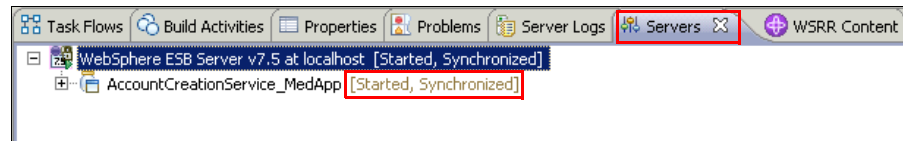


Figure 8-54 Application started and synchronized

Running the integrated test client

IBM Integration Designer comes with an Integration Test Client to test modules at development time. To start the Integration Test Client for the web service export ACS_WS_EXP from the mediation proxy module:

1. Right-click **ACS_WS_EXP** in the assembly diagram, and select **Test Component**, as shown in Figure 8-55.

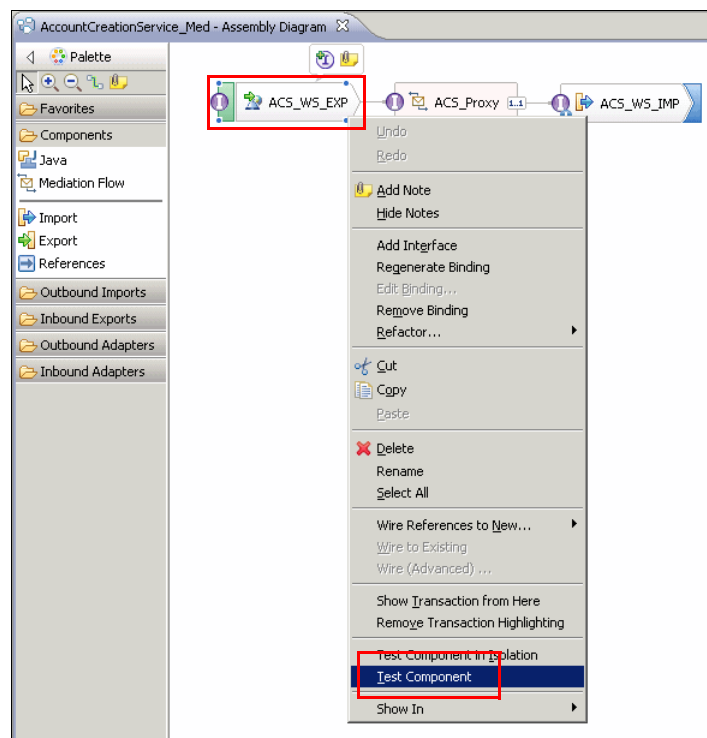


Figure 8-55 Start Integration Test Client for web service export ACS_WS_EXP

A new tab `AccountCreationService_Med_Test` opens. Review the test client and the operation that is called. On the right side, specify the test data that is used to create a test request against the chosen test component.

2. Insert the test data by right-clicking the **Envelope*** tag in the test data table.
3. Choose **Import from File** and navigate to the `Testdata_SOAPEnvelope_v1_0.xml` file.

Downloading the file: You can find the file `Testdata_SOAPEnvelope_v1_0.xml` in the additional material for this book. For information about how to download this material, refer to Appendix A, “Additional material” on page 395.

Alternatively, you can provide test data by manually setting the necessary attributes in the Integration Test Client. However, importing the test data is more convenient at this point.

Example 8-1 shows the input data for the SOAP envelope. You can see this by selecting **XML editor** in the Integration Test Client.

Example 8-1 Envelope data used as input for the message send to service endpoint

```
<?xml version="1.0" encoding="UTF-8"?><soap:Envelope
xmlns:ns1="http://SupplyCompany.itso.ibm.com/Account"
xmlns:ns3="http://SupplyCompany.itso.ibm.com/AccountCreationV1/interfac
e" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Header>
    <ns1:GEPHeaderB0 xsi:type="ns1:GEPHeaderB0">
      <consumerID>AMC100</consumerID>
      <contextID>AMC100CI01</contextID>
    </ns1:GEPHeaderB0>
  </soap:Header>
  <soap:Body>
    <ns3:createAccount>
      <customer>
        <creditLimit>10000</creditLimit>
        <active>false</active>
      </customer>
      <customer>
        <customerNumber>0815</customerNumber>
        <dateProfileCreated/>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <homePhoneNumber>0123456789</homePhoneNumber>
        <workPhoneNumber>+123456789</workPhoneNumber>
      </customer>
    </ns3:createAccount>
  </soap:Body>
</soap:Envelope>
```

```
<email/>
<dataValid>false</dataValid>
<Address>
  <street>John Doe Rd</street>
  <city>John Doe City</city>
  <state>John Doe State</state>
  <country>John Doe Country</country>
  <zip>00000</zip>
</Address>
</customer>
</customer>
</ns3:createAccount>
</soap:Body>
</soap:Envelope>
```

5. After starting the test client, you need to specify the deployment location for the application that is to be tested. As shown in Figure 8-57, select **WebSphere ESB Server v7.5 at localhost** as the deployment location, and click **Finish**.

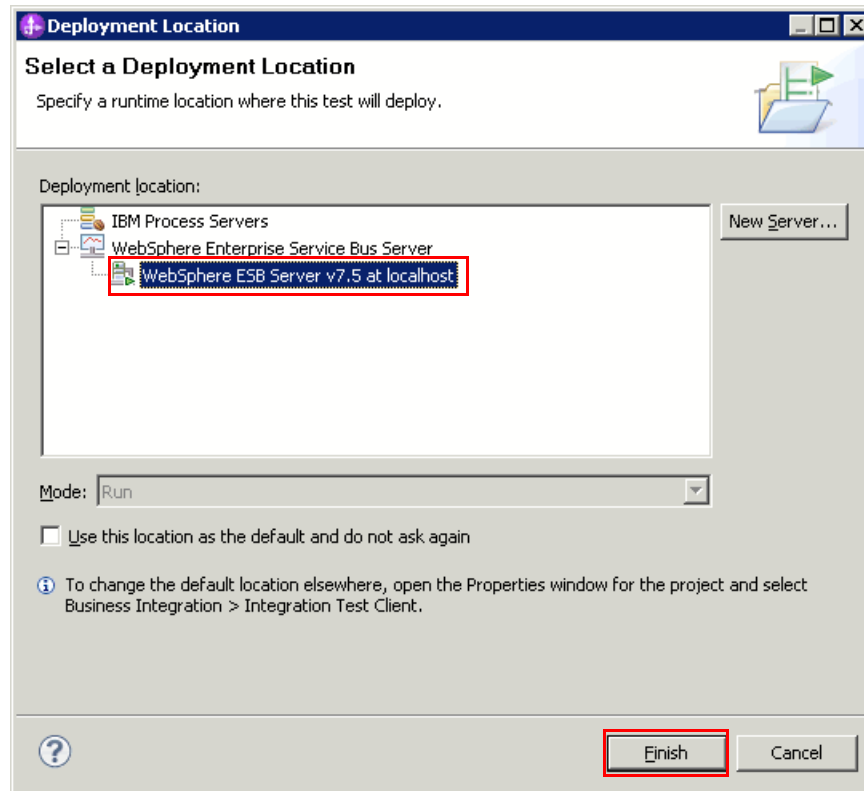


Figure 8-57 Select a deployment location

6. Provide the authentication credentials for the server as shown in Figure 8-58, and click **OK**.

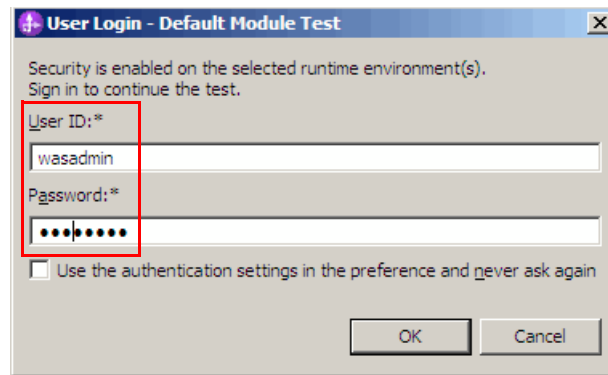


Figure 8-58 Authentication credentials for the server

7. After the test run is complete, the test client shows each test step and the corresponding test results, as shown in Figure 8-59.

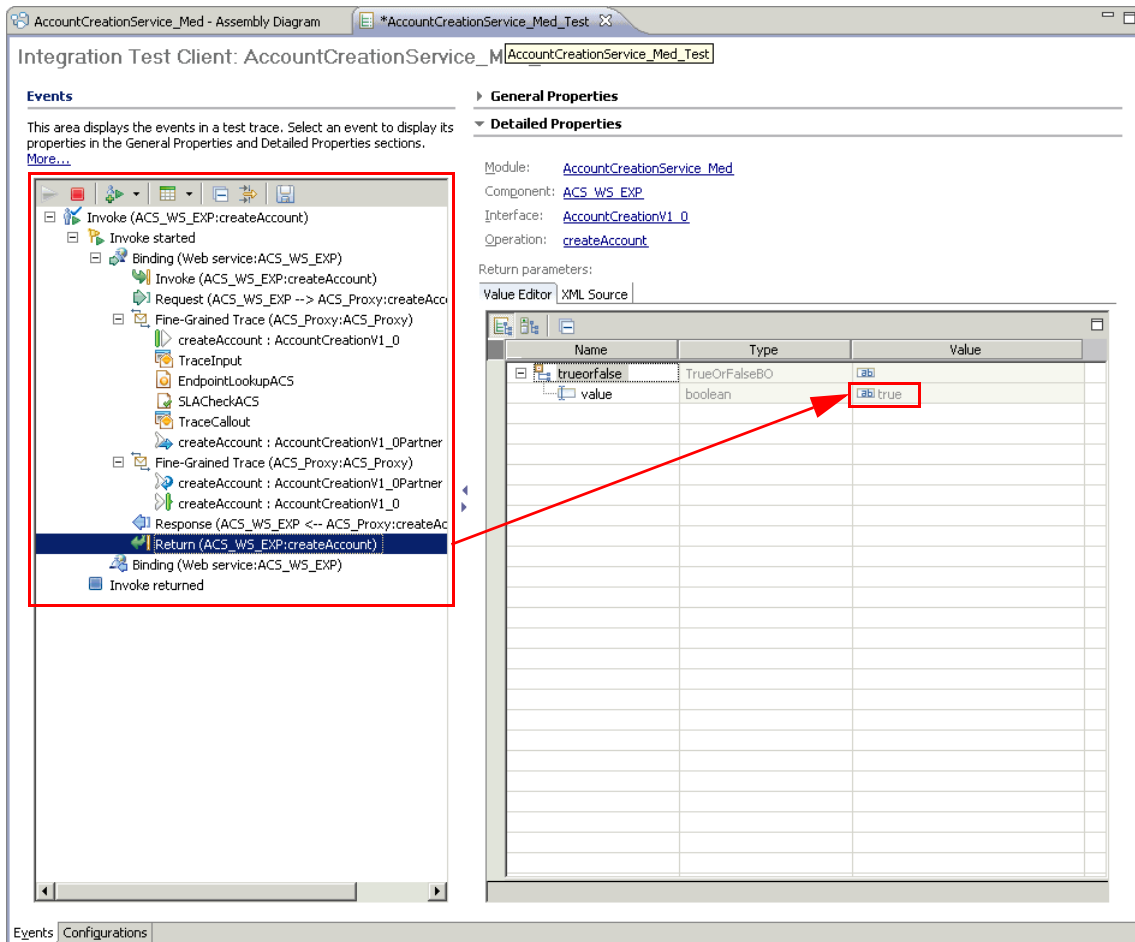


Figure 8-59 Completed test run

- Examine the test steps on the left and their corresponding variable states on the right. The test of the mediation proxy module AccountCreationService_Med was successful and the service provider AccountCreationV1_0EAR returned the value true.

Analyzing the results

In the next steps, we review the responses for the EndpointLookupACS mediation primitive, the SLACheckACS mediation primitive, the Callout node and the service provider AccountCreationV1_0EAR.

Figure 8-60 shows that the EndpointLookupACS mediation primitive connected successfully to the WSRR instance, and that it retrieves various information for the service AccountCreationV1_0.

The following endpoint is retrieved:

`https://localhost:9443/AccountCreationV1_0/services/AccountCreationServiceV1_0_DevelopmentPort`

In addition, the endpoint is written into the following message location:

`/headers/SMOHeader/Target/address`

The message is then forwarded to the SLACheckACS mediation primitive.

The screenshot shows the IBM WebSphere Integration Developer (WID) interface. On the left, the project structure is visible, with 'EndpointLookupACS' selected under the 'Fine-Grained Trace (ACS_Proxy:ACS_Proxy)' folder. A red arrow points from this selection to the 'Value Editor' on the right. The 'Value Editor' displays the 'mediationMessage' variable, showing a table of message properties.

Name	Type	Value
context	ContextType	
userContext	UserContext...	
transient	EObject	
primitiveContext	PrimitiveCont...	
dynamicProperty	DynamicProp...	
correlation	EObject	
failInfo	FailInfoType	
shared	EObject	
headers	HeadersType	
SOAPFaultInfo	SOAPFaultIn...	
MQHeader	MQHeaderType	
EISHeader	EISHeaderType	
SMOHeader	SMOHeaderT...	
Operation	String	createAccount
Target	TargetAddre...	
bindingType	BindingTypeT...	WebService
import	String	
repositoryMetadata	String	3079d930-e0c6-4632_a474.9fe0f39f74e1
address	String	https://localhost:9443/AccountCreati...V1_0
SourceNode	String	ACS_WS_EXP
MessageUUID	String	4532F134-012F-4000-E000-1190092AAB91
Interface	String	wsdl:http://SupplyCompany.itso.ibm.com/A...

Figure 8-60 EndpointLookupACS mediation primitive result

Figure 8-61 shows that after the SLACheckACS mediation primitive, the TraceCallout primitive is called. According to the implementation shown in Figure 8-47 on page 297, this primitive is wired to the accept terminal of the SLACheckACS mediation primitive.

Thus, an SLA between the caller, which is the test client that provides Consumer ID and Context ID, and the service provider AccountCreationV1_0EAR, exists in the registry. Therefore the message is forwarded to the Callout node through the accept terminal.

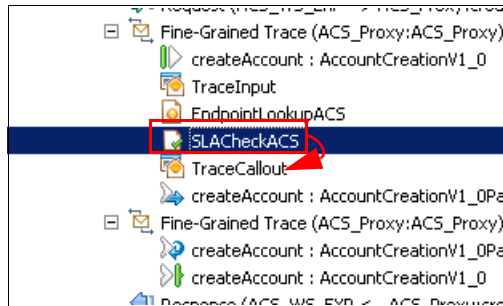


Figure 8-61 SLACheckACS mediation primitive result

As shown in Figure 8-62, the Callout node calls the service provider that is configured in the following message location:

/headers/SMOHeader/Target/address

Keep in mind that the service provider address was set previously by the EndpointLookupACS mediation primitive.

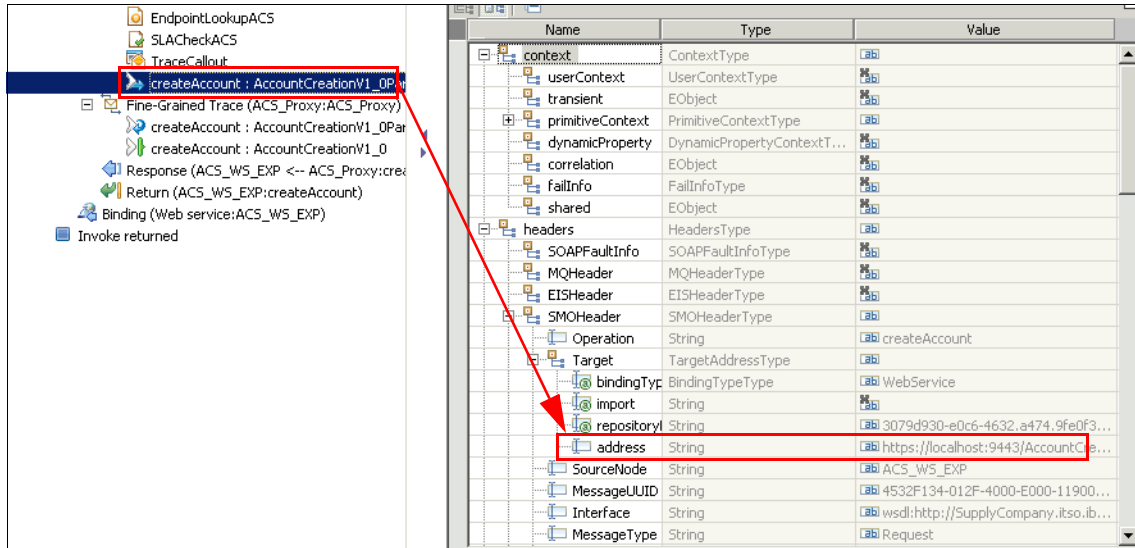


Figure 8-62 Callout node calls the endpoint set by EndpointLookupACS mediation primitive

The call to the service provider AccountCreationV1_0EAR was successful.

Example 8-2 shows the entries in the system console log written by the service implementation of AccountCreationV1_0EAR. We find the service name, the service version, and a statement including the customer number 0815 from the test data there, which proves that the service was invoked with a dynamically retrieved endpoint from WSRR.

Example 8-2 System console output for Account Creation Service

```
0000024c SystemOut      0 Service: Account Creation Service
0000024c SystemOut      0 Version: 1.0
0000024c SystemOut      0 Account for 0815 created successfully!
```

At this point, we have created a dynamic mediation proxy module with the basic capabilities for retrieving service endpoints and checking SLAs at run time. This module includes a mediation library module that stores the artifacts that are retrieved from WSRR, which are necessary to query for the service endpoint.

The mediation module now functions as a facade for all service consumers by isolating the service provider's implementation from the consumers and by providing a web service export on WebSphere ESB.



Extending the mediation

This chapter provides step-by-step instructions about how to extend the mediation flow that we describe in Chapter 8, “Implementing a mediation” on page 251. The changes that we introduce in this chapter provide a higher level of flexibility and dynamicity to the solution, which is needed to cover additional business requirements.

We also describe the benefits and the features that are provided by a second group of primitives, which offer integration capabilities between WebSphere Enterprise Service Bus (WebSphere ESB) and WebSphere Service Registry and Repository (WSRR).

9.1 Addressing the business requirements

We want to extend the flow that we describe in Chapter 8, “Implementing a mediation” on page 251 for several reasons. As the number of consumers of the service increases, technical improvements must be added to the mediation flow to provide the same level of quality to clients. In this scenario, new non-functional requirements are defined by the supply company, as described in Chapter 5, “The case study and scenario used in this book” on page 133. These requirements are the input for extending the mediation proxy.

We address the requirements as follows:

- ▶ Business requirements NFR-301 and NFR-302 aim to reduce interactions between the mediation module in WebSphere ESB and WSRR, and provide the same or a greater level of flexibility. Introducing an SLA Endpoint Lookup mediation primitive that replaces the Endpoint Lookup and SLA check mediation primitive by providing similar functionality satisfies this requirement. This improvement decreases the complexity of the mediation flow and provides additional filter capabilities for SLA and endpoint selection.
- ▶ Business requirement NFR-303 requires the mediation flow configuration to be changed dynamically at run time. The mediation module needs to be able to switch its mediation flow configuration based on the client’s country. By adding a Policy Resolution mediation primitive to the mediation flow, it enforces a policy that determines the flow behavior for a specific country at run time and satisfies this business requirement.

9.2 Other benefits of a mediation proxy module

In Chapter 8, “Implementing a mediation” on page 251, we discuss the benefits that are provided by a mediation proxy. In particular, we describe those benefits that are related to the following WebSphere ESB primitives, which interact with WSRR:

- ▶ Endpoint Lookup
- ▶ SLA Check

With these primitives, a greater dynamicity and flexibility is achieved. Now, we introduce the following mediation primitives and describe the benefits of using them:

- ▶ SLA Endpoint Lookup
- ▶ Policy Resolution
- ▶ Gateway Endpoint Lookup

With this second group of mediation primitives, many improvements are achieved such as implementing flows based on policy enforcement or more flexibility when querying the WSRR for endpoints or SLAs.

9.2.1 Benefits of the SLA Endpoint Lookup primitive

The SLA Endpoint Lookup primitive is an improvement of the Endpoint Lookup primitive. It combines into a single primitive the benefits described in 8.2.1, “Benefits of a dynamic endpoint lookup in a mediation” on page 255 and in 8.2.2, “Benefits of an SLA check in a mediation” on page 256. Using this primitive, the mediation flow is easier to implement and understand. In addition, performance can be improved because there is only one interaction between the mediation module (WebSphere ESB) and the registry.

This primitive also provides a customization mechanism, so that the behavior of the primitive can be adapted to support additional filters or even a different WSRR information model.

9.2.2 Benefits of the Policy Resolution primitive

Mediation policies provide the ability achieve greater levels of flexibility and administrative control over the mediation’s implementation, without changing the actual implementation itself. You can control service interactions dynamically, using context information.

Using mediation policies, the mediation flow dynamically configures service interactions and primitives. At run time, the values of module properties that are hard-coded in a mediation flow are replaced with the values that are defined in the mediation policy that matches a given selection criteria. This selection criteria can be based on the execution context or on the message content. Classification systems, such as the life cycle status, can also be used as part of the selection criteria.

A typical use case for mediation policies is when there is a business rule that indicates that the processing flow configuration depends on the type of consumer, and the consumer is part of the message content. Figure 9-1 illustrates this approach.

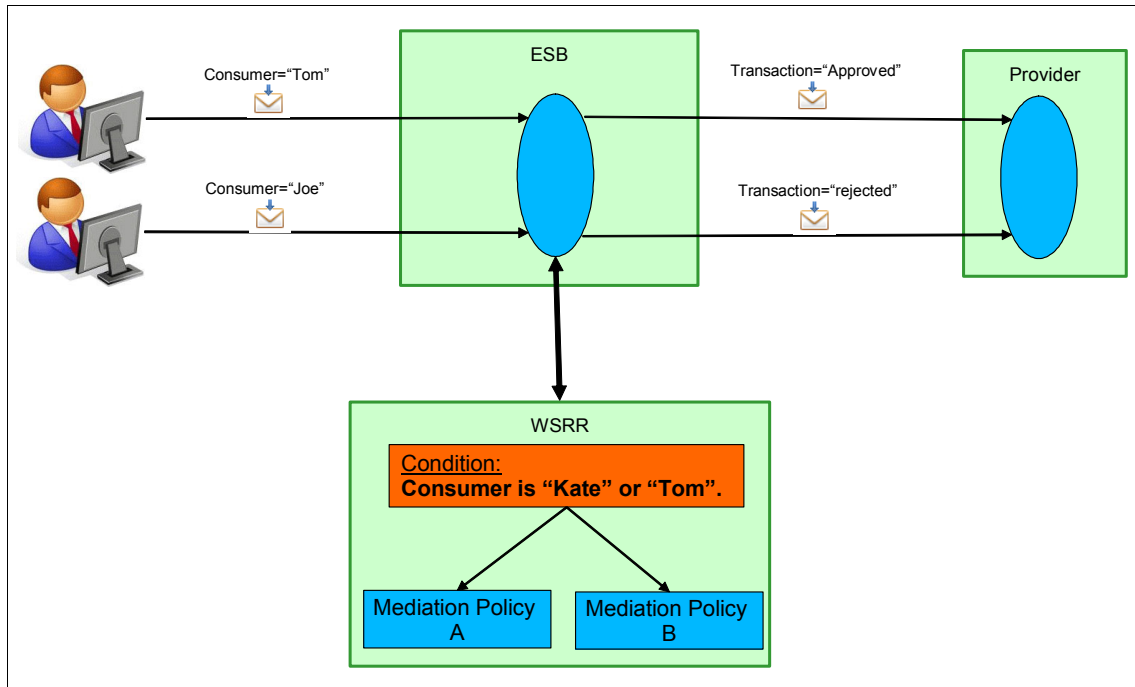


Figure 9-1 Policy Resolution mediation primitive using the message content

9.2.3 Benefits of the Gateway Endpoint Lookup primitive

This primitive, as its name suggest, is intended to retrieve and resolve endpoint information for routing purposes but implements a powerful gateway solution. The gateway, as an integration pattern, takes the message and, according to the service information that is stored in WSRR, routes the message to the final endpoint.

This primitive provides a function that uses the SOAPAction field or the WS-Addressing field provided in the SOAP message header as main endpoint lookup criteria. With the information stored in those fields, the primitive can query WSRR dynamically to determine the final endpoint to which to route the message.

This primitive is powerful because a generic gateway can be implemented to meet many non-functional requirements. A unique service endpoint can be exposed to consumers and, using this primitive, various non-functional controls such as security or logging can be centralized for a group of services. Figure 9-4 illustrates this approach.

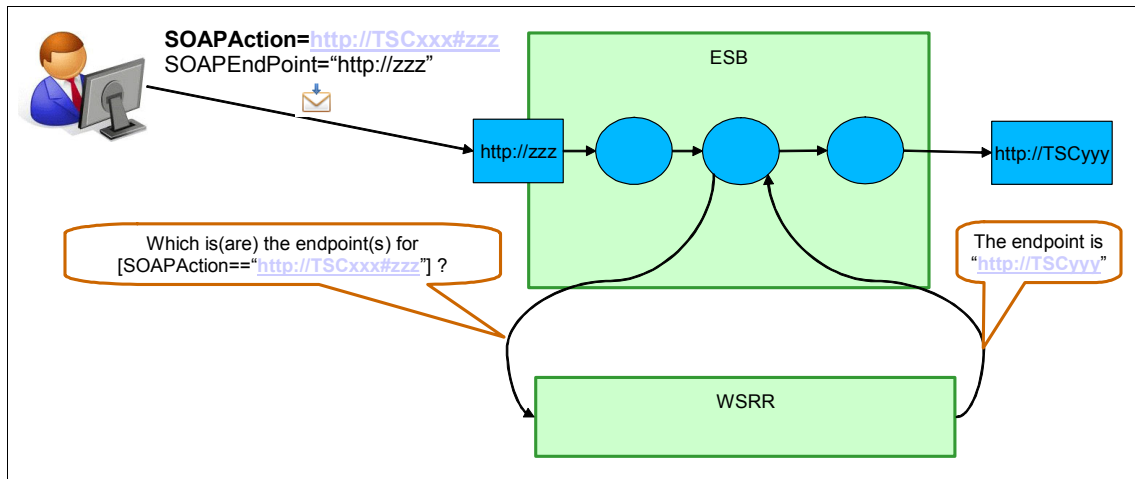


Figure 9-2 Gateway endpoint lookup using the registry

9.3 Additional primitives for WebSphere ESB Registry Edition

This section provides more detail about the second group of mediation primitives. Figure 9-3 shows these mediation primitives, which provide integration capabilities between WebSphere ESB and WSRR. We describe the EndpointLookup and SLA Check mediation primitives in Chapter 8, “Implementing a mediation” on page 251.

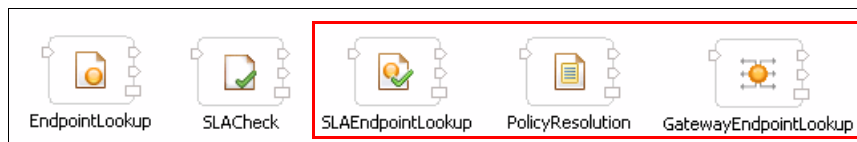


Figure 9-3 WESBRE advanced primitives

In this section, we focus first on the SLA Endpoint Lookup mediation primitive and the Policy Resolution mediation primitive, which are used to improve the

solution. Then, we describe the Gateway Endpoint Lookup mediation primitive, which implements a useful gateway pattern.

9.3.1 SLA Endpoint Lookup mediation primitive

The SLA Endpoint Lookup mediation primitive is used to look for available endpoints for an active SLA that has been agreed between a service consumer and a service provider. When using WebSphere ESB Registry Edition and the governance enablement profile, dynamic endpoints can be selected based on a number of factors that are modeled in the profile (governance enablement profile):

- ▶ Whether the consumer of the endpoint has a valid SLA for the endpoint
- ▶ Whether the particular SLA is active
- ▶ Whether the endpoint is online
- ▶ Whether the endpoint has a certain *Environment* classification, such as *Production* or *Development*

As shown in Figure 9-4, the SLA Endpoint Lookup mediation primitive has one input terminal, one fail terminal, and two output terminals (out and noMatch).



Figure 9-4 SLA Endpoint Lookup mediation primitive

The input terminal is wired to accept a message, and the other terminals are wired to propagate a message. If the information that is passed on the incoming message successfully finds a matching endpoint in WebSphere Service Registry and Repository, the out terminal is fired. However, if a matching endpoint is not found, the noMatch terminal is fired.

Several properties can be specified when configuring this primitive, as shown in Figure 9-5.

Figure 9-5 Configuration details for SLA Endpoint Lookup mediation primitive

To find a matching endpoint in WSRR, the SLA Endpoint Lookup mediation primitive uses information in the incoming message. The endpoint is matched on a number of parameters, which are defined by default as follows:

- Consumer Identifier

This field can be a literal value or can be passed as part of the message. It identifies the consumer of the target endpoint. The consumer identifier is a field in the Capability Version, Process Version, Application Version, or Service Version, representing the consumer of the target service. The SLA is referenced from this consumer entity by means of the Consumes relationship.

- Context Identifier

This field can be a literal value or can be passed as part of the message. It identifies the context under which the consumer's invocation of the target endpoint occurs. The context identifier is a field on the SLA, representing the agreement between the consumer and provider of the target service.

- Endpoint Classification

This field can be a literal value or can be passed as part of the message. It indicates a refinement in the selection of the target endpoint. All endpoints in WSRR can be classified. For example, they can be test endpoints or production endpoints. This classification enables a more fine-grained search.

You can add parameters into the search criteria to further refine the endpoint selection. Use the table of user-defined parameters under the Advanced tab, as shown in Figure 9-6, to supply extra selection parameters.

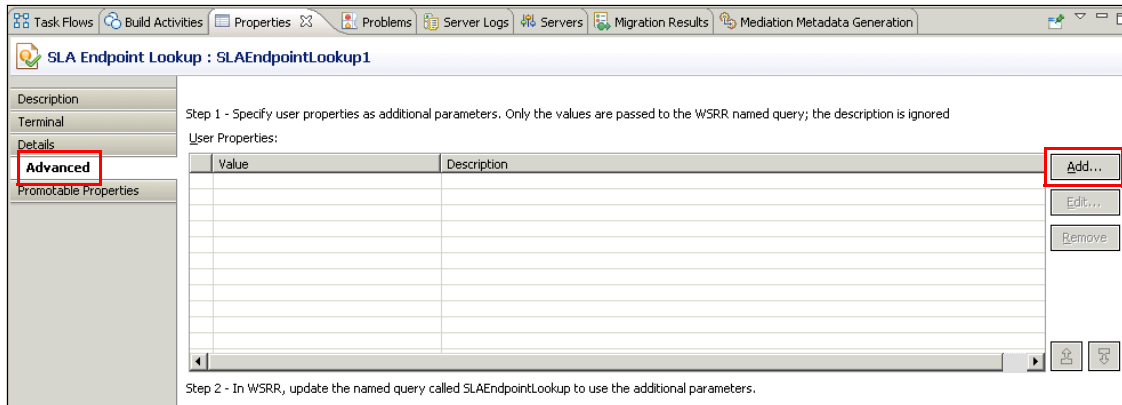


Figure 9-6 Advanced tab for SLA Endpoint lookup primitive

9.3.2 Policy Resolution mediation primitive

The Policy Resolution mediation primitive is used to retrieve and enforce mediation policies from WSRR and to configure at run time the mediation primitives that are executed later in the mediation flow. As shown in Figure 9-7, the Policy Resolution mediation primitive has one input terminal (in), two output terminals (out and policyError), and a fail terminal (fail).



Figure 9-7 Policy Resolution mediation primitive

The in terminal is wired to accept a message, and the other terminals are wired to propagate a message. If an exception occurs during the processing of the input message, the fail terminal propagates the original input message, together with any exception information. If no exceptions occur, the out terminal propagates the original service message object (SMO), which is modified by any property overrides that the mediation policies provided.

Note: There might not be any property overridden. The out terminal is also fired if there are no mediation policies to apply.

If valid mediation policies are found in the registry, their content is used to override dynamic properties of the mediation primitives that come after the Policy Resolution mediation primitive. If a mediation flow contains the Policy Resolution

mediation primitive, any promoted property that is in the top-level request, response, or fault flow is a dynamic property. Mediation policies contain the equivalent of promoted property groups, names, and values, and must be conform to the Web Services Policy Framework (WS-Policy).

At run time, when the Policy Resolution mediation primitive is executed, it runs a search query to the registry. You can specify whether the queries retrieve mediation policies that are associated with the module or its target services. The following policy scopes are possible:

Module	The Policy Resolution mediation primitive retrieves mediation policies that are attached to an SCA module object in WSRR.
Target service	<p>The Policy Resolution mediation primitive retrieves mediation policies that are attached to WSRR objects representing the target service. When you load a WSDL document into WSRR, the registry creates objects for any service, port, binding, portType, operation, and message element described in the WSDL. When you load an SCA module into WSRR, the registry creates objects for the module and any exports and imports defined in the module. The term <i>target service</i> encompasses all the relevant definitions included in a WSDL description and SCA exports.</p> <p>Generally, choose a point on which to attach mediation policies. For example, you might choose to attach policies to service objects or operation objects.</p>

Target service note: For this case, you must add an Endpoint Lookup primitive to the mediation flow in front of the Policy Resolution mediation primitive, so that the target service is known before trying to retrieve its mediation policy.

Intersection	The Policy Resolution mediation primitive retrieves mediation policies that are attached to both the module and the target service in WSRR. The Policy Resolution mediation primitive combines both scopes into a single mediation policy that meets the requirements of both. If the intersection processing cannot find a policy that meets both requirements, the <code>policyError</code> terminal is fired.
--------------	--

Intersection note: The run time supports mediation policies that are attached to service, port, binding, portType, and operation objects that are defined in WSDL documents. However, the run time does not support mediation policies that are attached to message objects.

For SCA modules, the run time supports mediation policies that are attached to Exports, the interface portType, and operations that are linked to the Export. The run time also supports mediation policies attached to the WSRR objects manual HTTP endpoint with associated interface, manual JMS endpoint with associated interface, and manual MQ endpoint with associated interface, and the interface portType and operations that are linked to these manual endpoints.

The properties settings for the Policy Resolution mediation primitive go beyond looking for a particular policy in a particular scope, as shown in Figure 9-8. The Conditions list allows you to specify context and content information that is going to be used for conditional policy resolution. Conditions allow different mediation policies to apply in different contexts. At run time, the conditions must be satisfied before a conditional mediation policy can be used.

These conditions are also referred to as *gate conditions*. If a gate condition resolves to true, the relevant mediation policy can be applied. A gate condition on a mediation policy is specified on a policy attachment within WSRR.

There is also a check box that indicates whether dynamic property values that are loaded by the Policy Resolution mediation primitive are visible for the response flow.

Task Flows Build Activities Properties Problems Server Logs Servers Migration Results Mediation Metadata Generation

Policy Resolution : PolicyResolution1

Description

Terminal

Details

Advanced

Promotable Properties

Registry Name: <Use default registry>

Policy Scope: Module

Conditions:

Policy condition name	XPath	Comment

Add... Edit... Remove

☐ Propagate mediation policy to response flow

Figure 9-8 Configuration details for Policy Resolution mediation primitive

When you design your mediation flow, any mediation primitive that occurs after the Policy Resolution mediation primitive can have its dynamic properties overridden by values from mediation policies. However, you must specify a valid default value for every property you want to override. Generally, you put a Policy Resolution mediation primitive at the beginning of a mediation flow.

You can use mediation policies in different ways. Here are two examples in which you can use mediation policies and function:

- ▶ Use mediation policies to activate or deactivate properties. For example, you can turn off message filtering by deselecting the Enabled property of the Message Filter primitive.
- ▶ Use conditional mediation policies, which apply when particular conditions exist. For example, you can apply different message transformations depending on different client types:
 - One transformation for gold clients
 - Another transformation for silver clients

The mediation policies can contain a different value for the XSL Transformation mapping file property, and depending on the client type, the appropriate mapping file is used.

Mediation policy processing model

The mediation policy processing model explains the conditions under which information is taken from mediation policies and applied to message flows. It also describes how the run time calculates the effective mediation policy for a particular scope, and when the run time applies intersection rules.

Impact on the SMO

If there are mediation policies that are attached to either the current module or to the target service, they are analyzed according to the mediation policy processing model. The resulting property information is copied to the SMO in the `/context/dynamicProperty` node. For each property, the `/context/dynamicProperty` node stores the group, name, and value. The property group and name are compared to dynamic properties in later mediation primitives. When a match occurs, the value of the dynamic property is overridden.

If you use mediation policies that are associated with target services, the run time matches the target service information in a particular message with the target service information in WSRR. For example, in WSRR, suppose you associate a mediation policy with the `getAddress` operation. At run time, the operation is taken from the SMO `/headers/SMOHeader/Operation` element.

9.3.3 Gateway Endpoint Lookup mediation primitive

The Gateway Endpoint Lookup mediation primitive is used to route service requests within a Service Gateway. There are three different modes, depending on the lookup method set in the primitive's properties:

- ▶ URL
- ▶ XPath
- ▶ Action

The URL and XPath modes are used when the module is acting as a proxy gateway and when endpoint lookups are based on a virtual service name. A proxy gateway can be created to route requests to virtual services that are defined in proxy groups. A proxy gateway is a specific type of service gateway in which an administrator can associate proxy groups with real endpoints. The service definitions are stored in a WebSphere ESB built-in configuration store.

The Action mode is used to resolve SOAP actions or WS-Addressing actions against WSRR for dynamic and static service gateway patterns. For example, a service gateway in Action mode can route to various web service endpoints, based on the action specified in the message. In contrast to service gateways in URL and XPath mode, the Action mode uses the service definitions that are stored in WSRR.

As shown in Figure 9-9, the Gateway Endpoint Lookup mediation primitive has one input terminal (in), two output terminals (out and noMatch) and a fail terminal (fail). The in terminal is wired to accept a message and the other terminals are wired to propagate a message.



Figure 9-9 Gateway Endpoint Lookup mediation primitive

If a suitable endpoint is found, the out terminal propagates the message and adds the endpoint to the SMO structure. If an endpoint is not found, the noMatch terminal is fired. You can wire the noMatch terminal to another mediation primitive that decides how to process the message. If an exception occurs during the endpoint lookup, the fail terminal propagates the original input message, together with any exception information contained in the failInfo element.

Properties

The properties settings for the Gateway Endpoint Lookup mediation primitive define which lookup method can be used and what proxy groups to include. Optionally if you select XPath as the lookup method, the Lookup XPath expression must be set, as shown in Figure 9-10.

Gateway Endpoint Lookup : GatewayEndpointLookup1

Description

Terminal

Details

Advanced

Promotable Properties

Lookup Method: URL

Proxy Group Names:

Proxy Group Name

Add...

Edit...

Remove

Lookup XPath:

Edit...

Registry Name: <Use default registry>

Figure 9-10 Configuration details for Gateway Endpoint Lookup mediation primitive

If the Gateway Endpoint Lookup mediation primitive finds suitable endpoints, it updates the SMO headers and context as follows:

- ▶ The dynamic callout address in the SMO header is set with the first match:
/headers/SMOHeader/Target/address
- ▶ The alternate targets list in the SMO header is updated with the remaining matches:
/headers/SMOHeader/AlternateTarget
- ▶ All services found are also placed in the primitive context:
/context/primitiveContext/EndpointLookupContext

bindingType note: The Gateway Endpoint Lookup mediation primitive also updates the bindingType associated with any endpoint addresses. In this case, the bindingType is always WebService.

The Gateway Endpoint Lookup mediation primitive uses the Endpoint Reference structure that is defined by the WS-Addressing specification. For more information, see:

<http://schemas.xmlsoap.org/ws/2004/08/addressing>

Action mode

The Gateway Endpoint Lookup mediation primitive, when configured in Action mode, uses the service definitions that are stored in WSRR to resolve the target endpoint for routing messages.

For a service gateway to use action-based routing, a web service request must contain an action either in the SOAPAction header or the WS-Addressing Action header. If a web service request contains both action headers, the values of the headers must be the same.

WSRR provides an action field that is based on each input and output defined in your WSDL. Using the input message, the Gateway Endpoint Lookup mediation primitive can query WSRR for all endpoints with the same action value. If necessary, you can specify classifications and user-defined properties that limit the search criteria, as shown in Figure 9-11.

The screenshot shows the 'Advanced' tab of the 'Gateway Endpoint Lookup : GatewayEndpointLookup1' configuration. The left sidebar contains tabs for 'Description', 'Terminal', 'Details', 'Advanced' (selected), and 'Promotable Properties'. The main area is divided into two sections, both highlighted with red rectangles:

- Classification URIs:** A text input field with 'Add...', 'Edit...', and 'Remove' buttons to its right.
- User Properties:** A table with columns 'Name', 'Type', and 'Value', followed by 'Add...', 'Edit...', and 'Remove' buttons.

Name	Type	Value

Figure 9-11 Gateway Endpoint Lookup mediation primitive Advanced tab

SOAP Note: Action-based lookups are only supported for SOAP 1.1.

9.4 Extending the mediation proxy

An extension of the mediation proxy is needed to meet the business requirements of the supply company. This section describes the tasks to add

enhancements to the solution that was implemented in 8.4, “Implementing a basic mediation proxy module” on page 264. We provide step-by-step instructions about how to implement and configure these enhancements using additional WebSphere ESB and WSRR integration features.

Prerequisites: For the procedures that we describe in this section, we assume that the environment is set up as follows:

- ▶ The runtime environment and the related tools are installed and configured, as described in Chapter 4, “Implementing a Low Workload topology” on page 77.
- ▶ The service registry instance contains the initial required information, such as WSDL documents and service metadata.

For your convenience, we provide a WSRR export file (WSRRExport001.zip) that contains all the required definitions. You can find it in the /Extend_Mediation/StartingPoint folder of the additional material file.

To import this file into WSRR, launch the WSRR console and in the Administrator perspective, select **Actions** -> **Import** and import WSRRExport001.zip.

- ▶ The Account Creation back-end service was deployed as explained in 8.4.3, “Testing the mediation proxy module” on page 298.

For your convenience, we provide the AccountCreationV1_OEAR.ear file, which contains the service implementation. You can find it in the /Extend_Mediation/StartingPoint folder of the additional material file.

- ▶ The Account Creation basic mediation was implemented, deployed, and tested, as described in 8.4, “Implementing a basic mediation proxy module” on page 264.

For your convenience, we provide an IBM Integration Designer project interchange file (ACS_PI_Basic.zip) that contains the basic mediation implementation. You can find it in the /Extend_Mediation/StartingPoint folder of the additional material file. To import a project interchange file into an IBM Integration Designer workspace, go to **File** → **Import**, and select **Project Interchange** under Others. Browse for the compressed file (.zip), and select the project that is contained in it.

9.4.1 Implementing an SLA-based endpoint resolution

In this section, we make use of the SLA Endpoint Lookup primitive. For a given set of query parameter values, this primitive looks in WSRR for existing SLAs, and it returns a list of the endpoints that are associated with those SLAs. We describe the benefits of using this primitive in 8.2.2, “Benefits of an SLA check in

a mediation” on page 256. Using this primitive in our existing solution provides the following advantages:

- ▶ We simplify the mediation flow implementation by replacing two existing primitives with this one, covering the same functionality.
- ▶ The SLA Endpoint Lookup primitive provides a customization mechanism so that the behavior of the primitive can be adapted to support additional filters. You can also customize the primitive to support a data model different from the one provided with the governance enablement profile.

Using the SLA Endpoint Lookup primitive

In this section, we use IBM Integration Designer to modify the existing mediation as follows:

1. Start IBM Integration Designer and open the workspace that contains the AccountCreationService_Med mediation project. Verify that you are in the Business Integration perspective.
2. In the Business Integration view, expand the **AccountCreationService_Med** project and double-click the **ACS_Proxy** mediation flow, shown in Figure 9-12, to open the mediation flow editor.

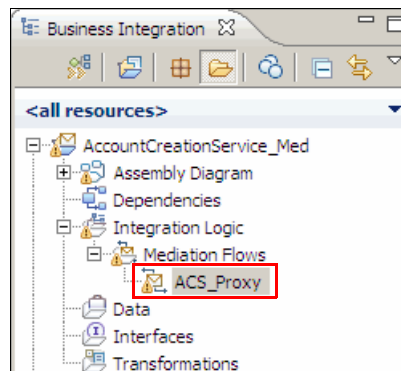


Figure 9-12 Browse project contents to open the ACS_Proxy mediation flow

In the Overview tab of the mediation flow editor, click the **createAccount** operation, as shown in Figure 9-13.

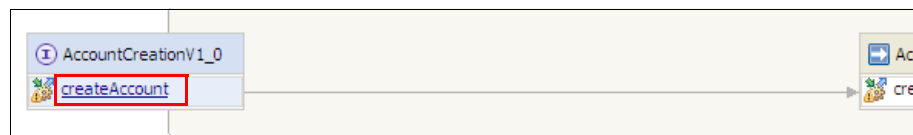


Figure 9-13 Mediation flow Overview tab, createAccount operation

Figure 9-14 shows the Request tab for the mediation flow that was implemented in 8.4, “Implementing a basic mediation proxy module” on page 264.

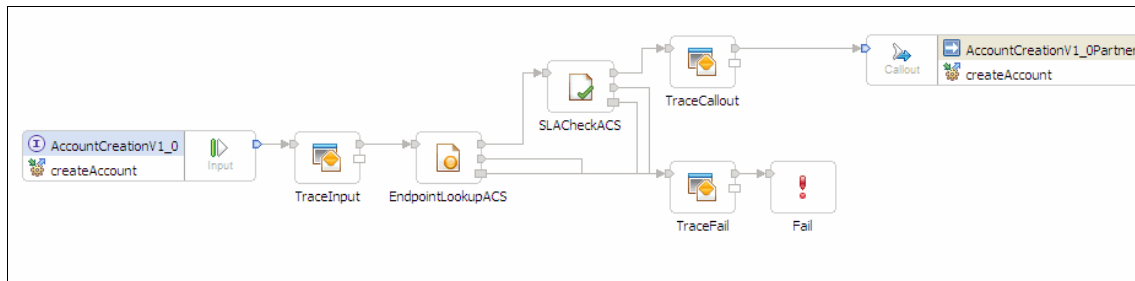


Figure 9-14 Initial request flow

3. Remove the `EndpointLookupACS` and `SLACheckACS` primitives from the editor flow.
4. Expand the **Routing** group in the mediation flow editor Palette and locate the **SLA Endpoint Lookup** primitive. Drag the primitive onto the flow editor. Right-click the new primitive, and then select **Rename**. Change the name to `SLAEndpointLookupACS`.
5. Wire the input and output terminals of the `SLAEndpointLookupACS` primitive as shown in Figure 9-15.

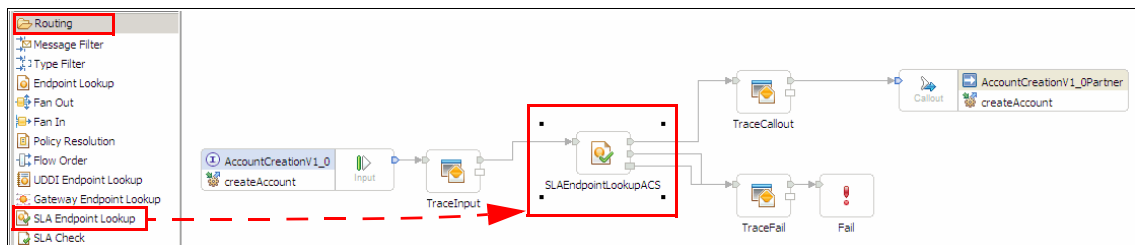


Figure 9-15 Modified request flow

6. Right-click `SLAEndpointLookupACS` and select **Show In -> Properties View**. Go to the **Details** tab of the Properties view. Configure the primitive according to the contents of WSRP, so that the primitive can retrieve the existing endpoints. Modify the properties as follows:
 - Consumer ID: XPath pointing to the SOAP header in the SMO
`/headers/SOAPHeader[name='GEPHeaderB0']/value/consumerID`
 - Context ID: XPath pointing to the SOAP header in the SMO
`/headers/SOAPHeader[name='GEPHeaderB0']/value/contextID`

- Endpoint Classification: String value

`http://www.ibm.com/xmlns/prod/serviceregistry/6/1/GovernanceProfileTaxonomy#Staging`

Verify that the property values are as shown in Figure 9-16, and save the modifications.

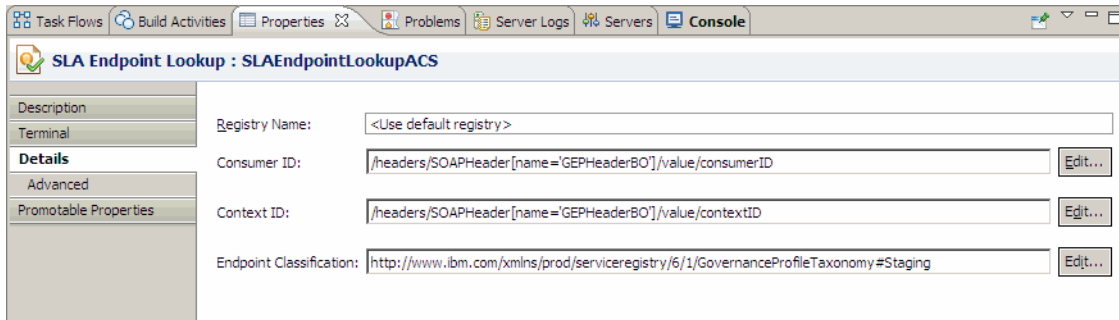


Figure 9-16 SLA Endpoint Lookup Details tab

7. At this point, the mediation flow is ready to deploy and test. Go to the Servers view, and verify that the server is running. Add the AccountCreationService_Med project to the server if it is not already added. Then right-click the server node and select **Publish** to redeploy the solution, as shown in Figure 9-17.

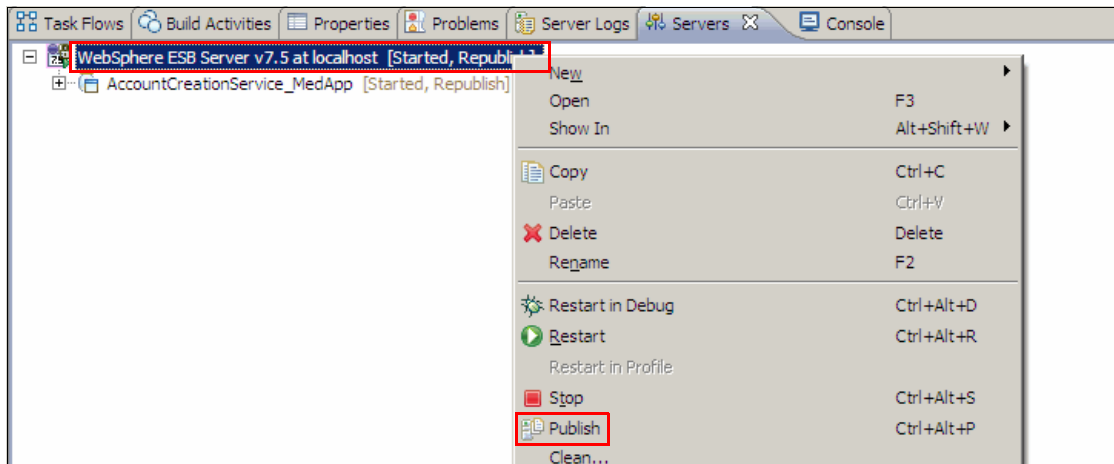


Figure 9-17 Publish changes to the server

Customizing the SLA Endpoint Lookup primitive

In this section, we describe how to customize the SLA Endpoint Lookup primitive to support an additional query parameter. In this case, we want to look for a specific Agreed Endpoint among those that are associated to the SLA definition in WSRR. We use the name property of the corresponding WSRR entity as the new query parameter. By doing this, we can support scenarios where multiple service level definitions are tied to an existing SLA, but a specific service level definition must be selected on a per request basis, depending on different conditions, for example the value of a request field.

Query Note: At run time, the SLA Endpoint Lookup primitive uses a predefined named query in WSRR. This query is parametrized. Thus, before running the query, the primitive replaces the query parameters with the values that are provided in the primitive configuration (Consumer ID, Context ID, and Classification). This query retrieves the endpoints that match these query parameters and that are online for an active SLA. In our scenario, we want to add a new parameter to this query.

To implement this scenario, follow these steps:

1. Modify the WSRR named query that is supporting the SLA Endpoint Lookup primitive. Open a browser window for the WSRR Console in the <https://localhost:9443/ServiceRegistry> address, and log in. Verify that you are in the Configuration perspective, as shown in Figure 9-18.

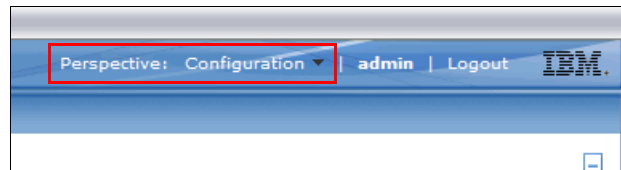


Figure 9-18 WSRR web application, configuration perspective

2. Go to **Active Profile** → **Named Queries** and click the **SLAEndpointLookup** link to open the query editor.

Make a backup: Before changing the named query, make a backup by clicking **Download Document** and save the file on your file system. For your convenience we provide the following folder, which contains both the original query and the customized named query:

/Extend_Mediation/Results/Part1 - SLAEndpointLookup/NamedQueries

3. Modify the query by adding the [*@name='%4'*] filter, as shown in Example 9-1. Click **OK** to save the changes.

Example 9-1 Customized SLAEndpointLookup named query

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- begin_generated_IBM_copyright_prolog -->

<!-- Licensed Materials - Property of IBM -->
<!-- 5724-N72 5655-W17 -->
<!-- (c) Copyright IBM Corp. 2010 All Rights Reserved -->
<!-- US Government Users Restricted Rights - Use, duplication or -->
<!-- disclosure restricted by GSA ADP Schedule Contract with -->
<!-- IBM Corp. -->

<!-- end_generated_IBM_copyright_prolog -->
<query
xmlns="http://www.ibm.com/xmlns/prod/serviceregistry/7/0/NamedQueryConfiguration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.ibm.com/xmlns/prod/serviceregistry/7/0/NamedQueryConfiguration ../schemas/NamedQueryConfiguration.xsd"
type="graphQuery">

<xpath>/WSRR/GenericObject[classifiedByAnyOf(.,'http://www.ibm.com/xmlns/prod/service
registry/profile/v6r3/GovernanceEnablementModel#CapabilityVersion') and
gep63_consumes(.) and
@gep63_consumerIdentifier='%1']/gep63_consumes(.)[classifiedByAnyOf(.,'http://www.ibm
.com/xmlns/prod/serviceregistry/lifecycle/v6r3/LifecycleDefinition#SLAActive') and
@gep63_contextIdentifier='%2']/gep63_agreedEndpoints(.)[@name='%4' 
/gep63_availableEndpoints(.)[classifiedByAnyOf(.,'http://www.ibm.com/xmlns/prod/servi
cereregistry/lifecycle/v6r3/LifecycleDefinition#Online') and
classifiedByAnyOf(.,'%3')]]</xpath>
    <depth>-1</depth>
</query>
```

4. Modify the mediation flow so that a new parameter is provided in the SLA Endpoint Lookup primitive. Go back to the mediation flow editor in IBM Integration Designer and select the **SLAEndpointLookupACS** primitive. In the **Advanced** tab of the Properties view, introduce a new User Property by clicking **Add**. Provide the following values, as shown in Figure 9-19:

Value	SLD - Account Creation Service
Description	SLD Name

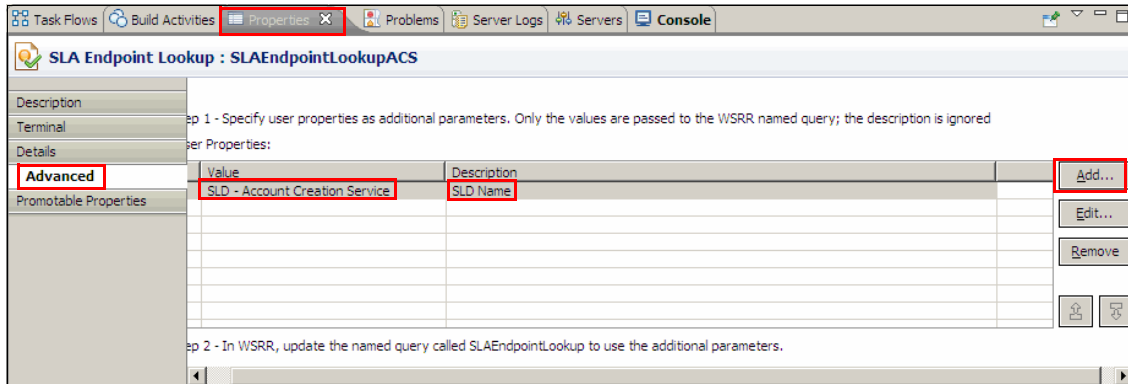


Figure 9-19 Advanced tab SLA Endpoint Lookup primitive properties

Note that the Value field contains the current name of the existing service level definition in WSRR. Use Ctrl+s to save the changes.

SLD Name parameter: For the purposes of this book, we set a fixed value for the SLD Name parameter. This scenario can be improved by adding dynamicity using the Policy Resolution Mediation primitive, as described in 9.4.2, “Implementing dynamic mediations” on page 336.

You can verify the name of the existing service level definition by using the WSRR web application. Switch to the SOA Governance perspective and click the **Service Level Definitions** link, which is available in the home page. The SLD - Account Creation Service entity is in the resulting list.

- At this point, the mediation flow is ready to deploy and test. Go to the Servers view. Verify that the server is running and that the AccountCreationService_Med project is added to the server. Right-click the server node and click **Publish** to redeploy the solution, as shown in Figure 9-17 on page 332.

After the server is in the Synchronized state, test the mediation as described in “Running the integrated test client” on page 304 and “Analyzing the results” on page 310. Verify that there are no errors when running the test.

Optional test: As an optional test, you can modify the value of the SLD Name user property (in the Advance tab of the Properties view for the SLAEndpointLookupACS primitive). Then, you can redeploy and run the test again. Set the SLD - Account Creation Service VIP for the user property value. Because this SLD is not in WSRR, this time the SLA Endpoint Lookup primitive will not retrieve any endpoint. The mediation reaches the Fail primitive, thus generating exception stack traces in the Console view.

Remember to restore the value of the SLD Name user property to SLD - Account Creation Service before continuing.

9.4.2 Implementing dynamic mediations

In this section, we make use of the Policy Resolution mediation primitive. By combining this primitive and the concept of promoted properties, we add an additional level of flexibility and dynamicity when building mediation flows in WebSphere ESB. In our scenario, using the Policy Resolution mediation primitive, we create and support two separate sets of promoted properties values (also called Mediation Policies). Each set is applied to the mediation flow on a “per request” basis, depending on the value of the parameter in the incoming message. At run time, these promoted properties dynamically change the behavior of the mediation flow, for example enabling or disabling a *login* or *tracing* primitive for conditional auditing.

To implement this enhancement you will perform the following:

- ▶ Configuring the Policy Resolution mediation primitive
- ▶ Registering the module in WSRR
- ▶ Creating mediation policy instances using Business Space

Configuring the Policy Resolution mediation primitive

In this section, we use IBM Integration Designer to implement the required changes in the mediation flow. To take advantage of the Policy Resolution mediation primitive:

1. Return to IBM Integration Designer and verify that the request flow for the ACS_proxy mediation is open. In the request flow of the ACS_Proxy mediation, right-click **TraceInput** and select **Rename**. Change the name to AuditInputACS.

Message auditing: Message auditing is a common requirement in mediation modules. WebSphere ESB provides specific primitives for this functionality, such as the Message Logger primitive or the Event Emitter primitive. Exploring this topic is beyond the scope of this book so for simplicity, we use one of the existing Trace primitives, for illustrative purposes only.

2. In the mediation flow editor palette expand the **Routing** group and locate the **Policy Resolution** primitive. Drag the primitive onto the flow editor. Right-click the new primitive and select **Rename**. Change the name to `PolicyResolutionACS`. Wire the primitive as shown in Figure 9-20.

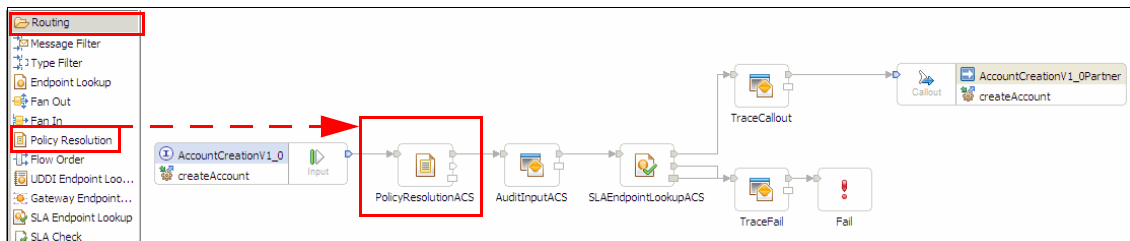


Figure 9-20 ACS_proxy request flow including the Policy Resolution Mediation primitive

3. Right-click **PolicyResolutionACS** and select **Show In -> Properties View**. Select the **Details** tab of the Properties view. Create a policy selection filter (also called *Policy Condition*) that is based on the value of the “country” field of the incoming message. By doing this, WebSphere ESB can apply different mediation policies, depending on the value of that field at run time. Click **Add** and provide the property values as shown in Figure 9-21. For the XPath property, you can type the XPath expression or use the Edit button to browse on the SMO and select the corresponding node.
 - Policy Condition Name: Country
 - XPath: `/body/createAccount/customer/customer/Address/country`

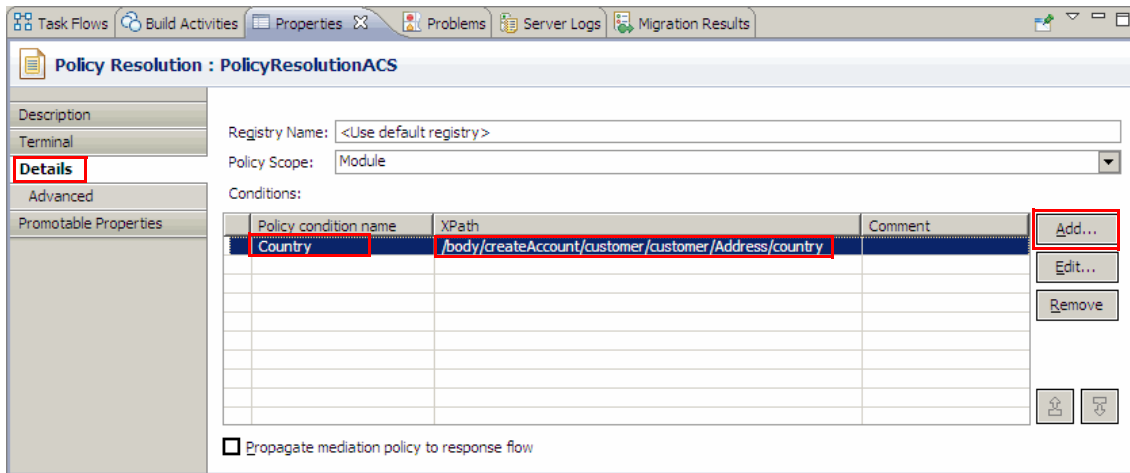


Figure 9-21 Policy Resolution Mediation primitive Details tab

- Click a blank empty area in the request flow editor and select the **Promotable Properties** tab in the Properties view. A list with all the available mediation properties displays. Select the check boxes in the Promoted column for the elements that are included in Table 9-1.

Table 9-1 Promotable properties

Primitive	Property
createAccount:AccountCreationV1_0Partner	Retry count
AuditInputACS	Enabled
	Message
TraceCallout	Enabled
TraceFail	Enabled
SLAEndpointLookupACS	SLD Name [Value]

- Set the Alias value for the SLD Name [Value] property to SLAEndpointLookupACS.SLDName as shown in Figure 9-22.

Primitive	Property	Promoted	Group	Alias	Alias value
createAccount : Accou...	Automatically convert the ...	<input type="checkbox"/>			
createAccount : Accou...	Use dynamic endpoint if se...	<input type="checkbox"/>			
createAccount : Accou...	Async timeout (seconds)	<input type="checkbox"/>			
createAccount : Accou...	Invocation Style	<input type="checkbox"/>			
createAccount : Accou...	Retry on	<input type="checkbox"/>			
createAccount : Accou...	Retry count	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	createAccount_Callout.retryCount	3
createAccount : Accou...	Retry delay (seconds)	<input type="checkbox"/>			
createAccount : Accou...	Try alternate endpoints	<input type="checkbox"/>			
AuditInputACS	Enabled	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	AuditInputACS.enabled	true
AuditInputACS	Destination	<input type="checkbox"/>			
AuditInputACS	File path	<input type="checkbox"/>			
AuditInputACS	Message	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	AuditInputACS.literal	{0}, {1}, {2}, {3}, {4}, {5}
AuditInputACS	Root path	<input checked="" type="checkbox"/>			
TraceCallout	Enabled	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	TraceCallout.enabled	true
TraceCallout	Destination	<input type="checkbox"/>			
TraceCallout	File path	<input type="checkbox"/>			
TraceCallout	Message	<input type="checkbox"/>			
TraceCallout	Root path	<input type="checkbox"/>			
Fail	Error message	<input type="checkbox"/>			
Fail	Root path	<input type="checkbox"/>			
TraceFail	Enabled	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	TraceFail.enabled	true
TraceFail	Destination	<input type="checkbox"/>			
TraceFail	File path	<input type="checkbox"/>			
TraceFail	Message	<input type="checkbox"/>			
TraceFail	Root path	<input type="checkbox"/>			
SLAEndpoint.lookupACS	Registry Name	<input type="checkbox"/>			
SLAEndpoint.lookupACS	Consumer ID	<input type="checkbox"/>			
SLAEndpoint.lookupACS	Context ID	<input type="checkbox"/>			
SLAEndpoint.lookupACS	Endpoint Classification	<input type="checkbox"/>			
SLAEndpoint.lookupACS	SLD Name [Value]	<input checked="" type="checkbox"/>	AccountCreationService_Med.ACS_Proxy	SLAEndpoint.lookupACS.SLDName	SLD - Account Creation Service
PolicyResolutionACS	Policy Scope	<input type="checkbox"/>			
PolicyResolutionACS	Propagate mediation policy...	<input type="checkbox"/>			

Figure 9-22 Mediation flow Promotable Properties tab

6. Save the changes to the mediation.

Registering the module in WSRR

The Policy Resolution Mediation primitive relies on the information that a WSRR instance holds for the mediation module in which the primitive was introduced. At the same time, when registering a mediation module in WSRR, additional policy metadata information is added. A new policy domain is created to support the promoted properties that were defined in the module, so that it is possible to create policy mediation instances for that module.

To register the mediation module:

1. In the Business Integration view of IBM Integration Designer, right-click the AccountCreationService_Med node and click **Export**, as shown in Figure 9-23.

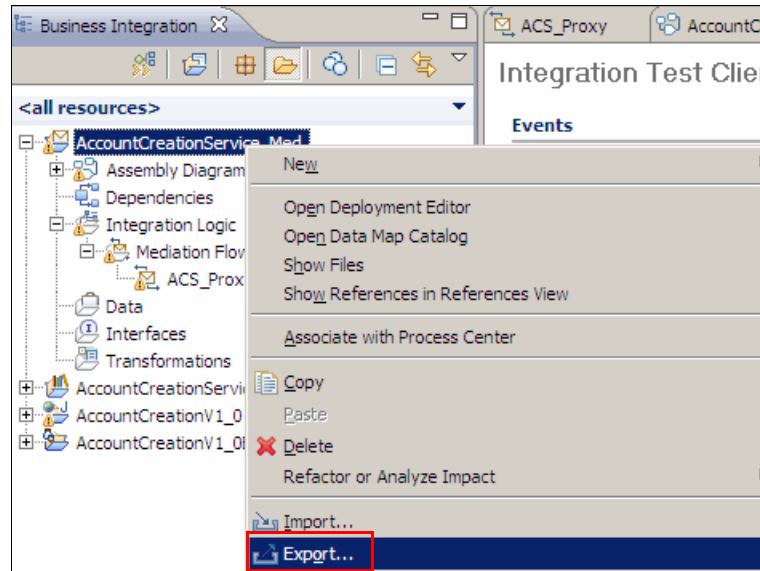


Figure 9-23 Exporting a mediation module

2. In the Select export destination dialog box, choose **EAR file** under the Java EE category and click **Next**. A new dialog box prompts you for a Destination folder. Click **Browse** and select a folder. Verify that AccountCreationService_Med is selected and click **Finish**.
3. Launch the WSRR web application in a web browser. Use the URL <https://localhost:9443/ServiceRegistry> and log in using a valid credential.
4. Verify that you are in the **Administrator** perspective. Go to **Actions** → **Load Documents**. The Load Documents form opens, as shown in Figure 9-24. Click **Browse** and select the EAR file containing your mediation module (AccountCreationService_MedApp.ear). Set Document type to **SCA integration module** and click **OK**.

After a few seconds, a message informs you of the successful publication of the mediation module.

Load Documents

This facility enables you to load one or more documents, with the option to save them as a group. Specify a file to load, select a document type and, optionally, enter a description and a version.

Path to the Document

☒ Local file system

Specify path
C:\Documents and Settings\Administrator\Desktop\AccountCrea

☐ Remote file location

Specify URL

Document type

SCA integration module

Enter document description:

Enter document version:
1.0.0

Figure 9-24 Load documents in WSRR web console

Creating mediation policy instances using Business Space

In this section, we use Business Space to create and attach two different mediation policies to our mediation module. The first policy is the default policy. The second policy is a specific policy and is applied only for requests where the “country” field has the value USA. For each policy, we set values for the promoted properties. For the second policy, we specify the conditions that are used to select the policy at run time.

To create the mediation policies:

1. Open a browser window and launch Business Space. We used the URL <https://localhost:9443/BusinessSpace> address. Log in as an administrator.
2. Click **Go to Spaces**. A list with the available spaces opens, as shown in Figure 9-25. Click **Solution Administration** and wait a few seconds while the space is loaded in your browser.

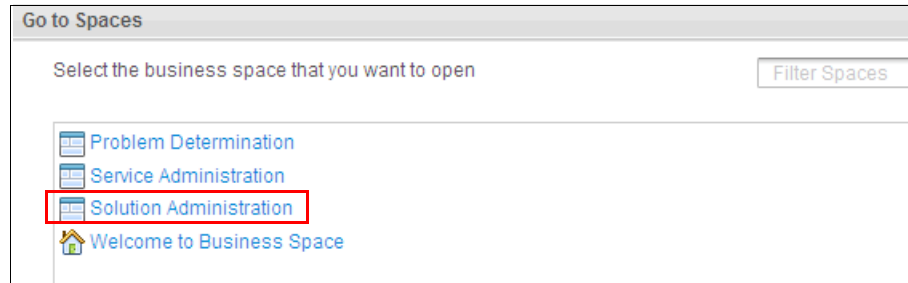


Figure 9-25 Go to Spaces dialog box: Business Space portal

3. The Solution Administration space will open, as shown in Figure 9-26. In the Module Browser widget, expand the **AccountCreationService_Med** node. Then, click **Module Policies** to load the mediation policy editor.

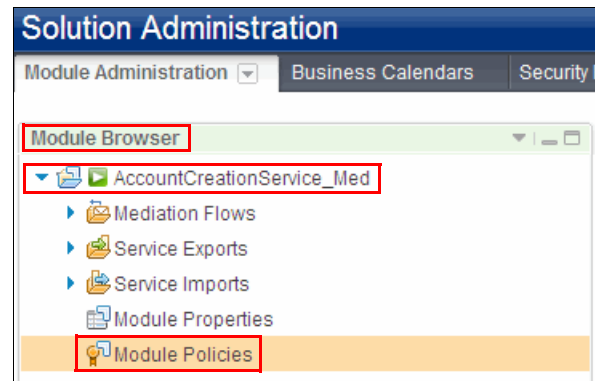


Figure 9-26 Opening the mediation policies editor

4. The mediation policy editor, shown in Figure 9-27, is loaded on the Module Administration widget. Create the first mediation policy by entering ACSMed_defaultConfig in the New policy attachment field, and click **Create**.

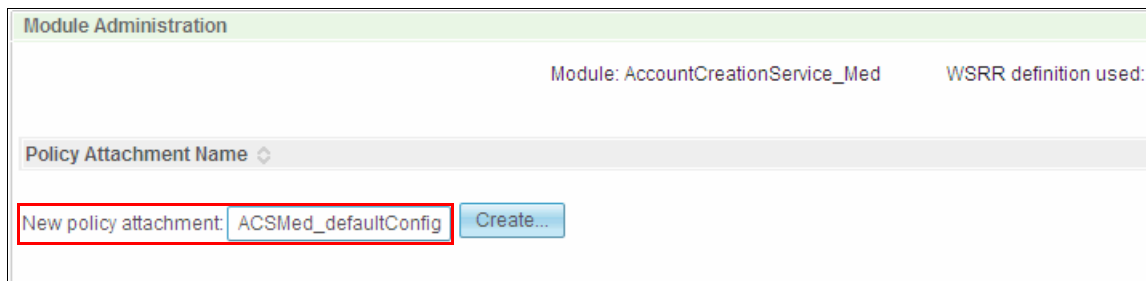


Figure 9-27 Mediation policy editor - Create new policy attachment

5. A list with the available group names for promoted properties displays, as shown in Figure 9-28. Click **AccountCreationService_Med.ACS_Proxy**, and select **Create New**. Enter `ACSMed_defaultPolicy` in the New policy field, and click **Next**.

Note: When you publish the mediation module in WSRR, each group of promoted properties in IBM Integration Designer becomes a “Mediation Policy Template” in WSRR.

Group Name Selection

Group Name ▾

AccountCreationService_Med.ACS_Proxy

Policy Selection

☐ Use existing

Select a policy ▾

☒ Create new

New policy: ACSMed_defaultPolicy

Figure 9-28 Select promoted properties group for creating the mediation policy

6. A new panel with the Assertions section and the Gate Conditions section opens. Under the Assertions section, shown in Figure 9-29, select **TraceFile.enabled** in the Property name field. Enter `false` for the Value field, and click **Add assertion**. By doing this we provide the value that, at run time, is set for the corresponding mediation property.

Assertions

Group Name ▾	Property Name ▾	Value ▾
Group name: AccountCreationService_Med.ACS_Proxy	TraceFail.enabled	false

Add Assertion

Figure 9-29 Adding a new assertion for the mediation policy.

Repeat this step for the remaining properties, and apply the values listed in Table 9-2.

Table 9-2 Assertions for ACSMed_USPolicy

Property name	Value
TraceFail.enabled	false
TraceCallout.enabled	false
AuditInputACS.enabled	true
AuditInputACS.literal	Message {1} audited at {0}
SLAEndpointLookupACS.SLDName	SLD - Account Creation Service
createAccount_Callout.retryCount	2

7. The default mediation policy is configured and ready to be saved as shown in Figure 9-30. Review the assertions, and click **Save** to complete the task.

Policy: ACSMed_defaultConfig

Assertions

Group Name	Property Name	Value
AccountCreationService_Med.ACS_Proxy	TraceFail.enabled	false
AccountCreationService_Med.ACS_Proxy	TraceCallout.enabled	false
AccountCreationService_Med.ACS_Proxy	AuditInputACS.enabled	true
AccountCreationService_Med.ACS_Proxy	createAccount_Callout.retryCount	2
AccountCreationService_Med.ACS_Proxy	AuditInputACS.literal	Message {1} audited at {0}
AccountCreationService_Med.ACS_Proxy	SLAEndpointLookupACS.SLDName	SLD - Account Creation Service

Group name: AccountCreationService_Med.ACS_Proxy Property name: Value:

Gate Conditions (Optional)

Name Value

Gate condition name: medGate_ Value:

Figure 9-30 List of created assertions for ACSMed_defaultPolicy

The ACSMed_defaultPolicy is created and attached to the mediation module without gate conditions, acting as a default setter for the promoted properties at run time. In the next steps, we create a second mediation policy with a gate condition.

8. Repeat steps 3 to 5, using the following values:
- New policy attachment name: ACSMed_USConfig
 - New policy name: ACSMed_USPolicy
 - Policy assertions are shown in Table 9-3.

Table 9-3 Assertions for ACSMed_USPolicy

Property name	Value
TraceFail.enabled	false
TraceCallout.enabled	false
AuditInputACS.enabled	false
AuditInputACS.literal	Message {1} audited at {0}
SLAEndpointLookupACS.SLDName	SLD - Account Creation Service
createAccount_Callout.retryCount	3

- Under the Gate Conditions section, shown in Figure 9-31, enter Condition1 for the condition name and Country = 'USA' in the condition value. Click **Add Gate Condition** to confirm the gate condition. By doing this, you specify that this policy is applied only for those requests where the field “country” has the value USA. In this case, we use the condition name “Country,” which at run time takes the value of the /body/. . . . /country XPath expression, set previously in the mediation primitive.

Gate Conditions (Optional)

Name	Value
Gate condition name: medGate_ Condition1	Value: Country = 'USA'

Add Gate Condition

Figure 9-31 Adding a new gate condition for the mediation policy

- The second mediation policy is configured and ready to be saved as shown in Figure 9-32. Review the assertions and the gate condition, and click **Save** to complete the task.

Group Name	Property Name	Value
AccountCreationService_Med.ACS_Proxy	SLAEndpointLookupACS.SLDName	SLD - Account Creation Service
AccountCreationService_Med.ACS_Proxy	AuditInputACS.literal	Message {1} audited at {0}
AccountCreationService_Med.ACS_Proxy	createAccount_Callout.retryCount	3
AccountCreationService_Med.ACS_Proxy	AuditInputACS.enabled	false
AccountCreationService_Med.ACS_Proxy	TraceFail.enabled	false
AccountCreationService_Med.ACS_Proxy	TraceCallout.enabled	false

Group name: AccountCreationService_Med.ACS_Proxy Property name: Value:

Gate Conditions (Optional)

Name	Value
medGate_Condition1	Country = 'USA'

Gate condition name: medGate_ Value:

Figure 9-32 List of created assertions for ACSMed_USPolicy

9.4.3 Testing the solution

This section describes how to test the changes that we introduced with the Policy Resolution mediation primitive. Partial tests related to the SLA Endpoint Lookup primitive were already completed in 9.4.1, “Implementing an SLA-based endpoint resolution” on page 329.

To test the solution, follow these steps:

1. In IBM Integration Designer, go to the Servers view and verify that the server is running and that the AccountCreationService_Med project is added to the server. Then right-click the server node and click **Publish** to redeploy the solution, as shown in Figure 9-33.

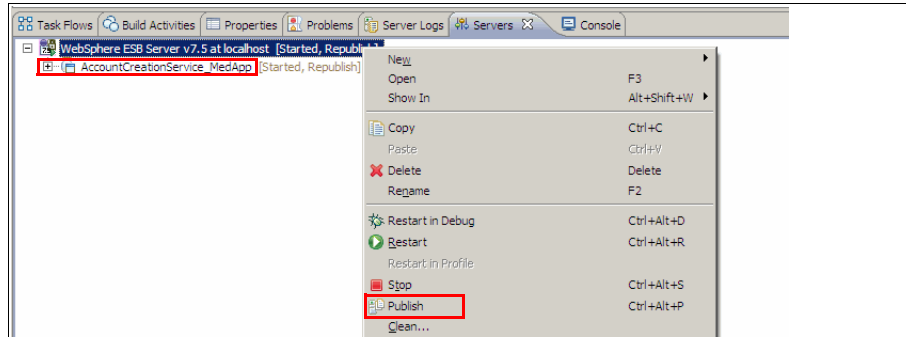


Figure 9-33 Publish changes to the server

2. After the server is in the Synchronized state, return to the Integration Test Client and click the **Invoke** and **Continue** icons to launch another test. Verify that there are no errors when running the test. The PolicyResolutionACS primitive was executed as part of the request flow, and the returned response from the back-end service is true, as shown in the Value Editor tab. See Figure 9-34.

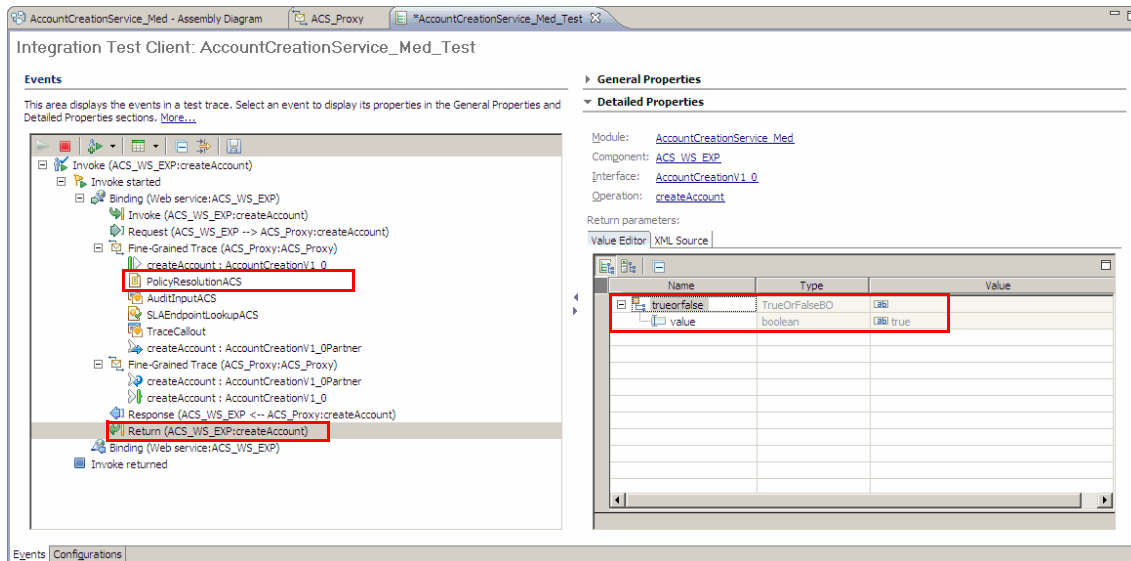


Figure 9-34 Integration Test Client view: Successful execution

3. Select the **PolicyResolutionACS** node in the Events tree. In the Value Editor tree, go to the **context** → **dynamicProperty** → **propertySets** node and expand all the properties child nodes, as shown in Figure 9-35. You can verify the names and values for each promoted property. In this case, the values

were loaded by the PolicyResolutionACS primitive, and they are according to the ACSMed_defaultPolicy mediation policy that we created previously using Business Space.

Name	Type	Value
context	ContextType	[ab]
userContext	UserContextType	[ab]
transient	EObject	[ab]
primitiveContext	PrimitiveContextType	[ab]
dynamicProperty	DynamicPropertyContextType	[ab]
propertySets	DynamicPropertySetType[] <Dyna...	[ab]
propertySets[0]	DynamicPropertySetType	[ab]
properties	PropertyType[] <PropertyType>	[ab]
properties[0]	PropertyType	[ab]
name	String	[ab] TraceCallout.enabled
value	String	[ab] false
type	String	[ab]
properties[1]	PropertyType	[ab]
name	String	[ab] AuditInputACS.enabled
value	String	[ab] true
type	String	[ab]
properties[2]	PropertyType	[ab]
name	String	[ab] AuditInputACS.literal
value	String	[ab] Message {1} audited at {0}
type	String	[ab]
properties[3]	PropertyType	[ab]
name	String	[ab] TraceFail.enabled
value	String	[ab] false
type	String	[ab]
properties[4]	PropertyType	[ab]
name	String	[ab] createAccount_Callout.retryCount
value	String	[ab] 2
type	String	[ab]
properties[5]	PropertyType	[ab]
name	String	[ab] SLAEndpointLookupACS.SLDName
value	String	[ab] SLD - Account Creation Service
type	String	[ab]
group	String	[ab] AccountCreationService_Med.ACS_Proxy

Figure 9-35 Promoted properties loaded from the “default” mediation policy

4. If we take a look at the Console view, shown in Figure 9-36, we can verify that there is a TraceMediation line that corresponds to the execution of the AuditInputACS primitive. The figure also shows the SystemOut lines that correspond to the execution of the back-end service.

Timestamp	Level	Source	Message
[4/18/11 15:36:33:473 EDT]	0000007c	ExportRuntime	I processMessage
[4/18/11 15:36:33:520 EDT]	0000007c	WSChannelFram	A CHFW0019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:6
[4/18/11 15:36:34:379 EDT]	0000007c	TraceMediatio	I CWSXMS010I: Message 6A1D43E0-012F-4000-E000-041CC0A89188 audited at 4/18/11 3:36 PM
[4/18/11 15:36:36:613 EDT]	0000007c	WSChannelFram	A CHFW0019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:6
[4/18/11 15:36:36:629 EDT]	00000076	SystemOut	O Service: Account Creation Service
[4/18/11 15:36:36:629 EDT]	00000076	SystemOut	O Version: 1.0
[4/18/11 15:36:36:629 EDT]	00000076	SystemOut	O Account for 0815 created successfully!
[4/18/11 15:36:36:660 EDT]	0000007c	ExportRuntime	I processMessage

Figure 9-36 Console view with mediation audit and back-end service traces

5. Execute the test again, but this time change the value in the request message for the country element to USA. In the Integration Test Client view, click the **Invoke** button, as shown in Figure 9-37.



Figure 9-37 Invoke button: Integration Test Client

A new Invoke node is created in the Events tree. In the Message - Value Editor tab, expand the request node and set the value for the /body/createAccount/customer/customer/Address/country field to USA, as shown in Figure 9-38.

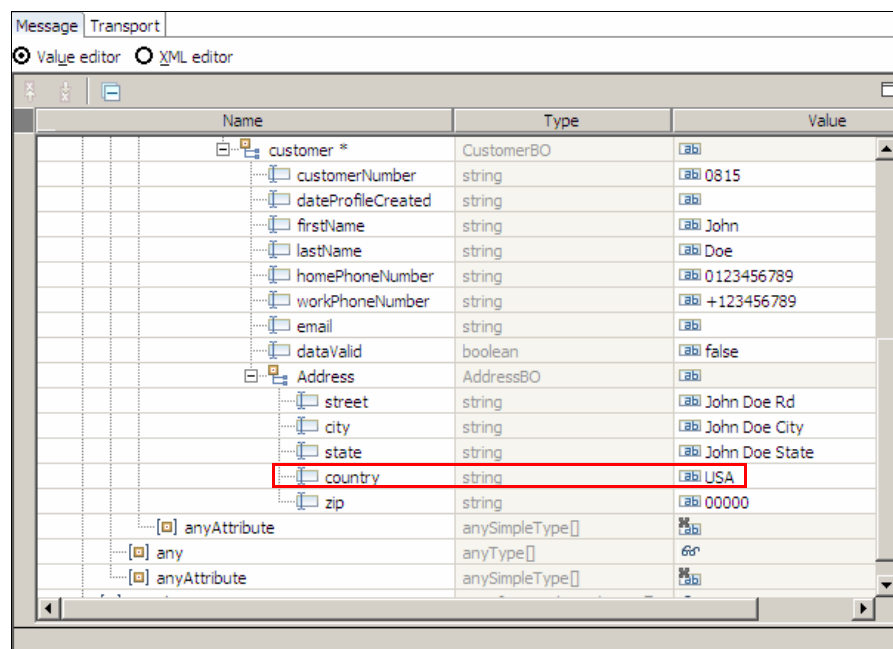


Figure 9-38 Changing the “country” for the new request

6. Start the execution of the new test by clicking **Continue**, shown in Figure 9-39.



Figure 9-39 Continue button, Integration Test Client

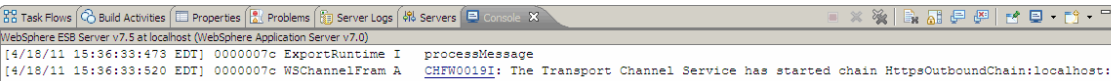
- After the execution has completed, select the **PolicyResolutionACS** node in the Events tree. Review again the values for the promoted properties, as shown in Figure 9-40.

Name	Type	Value
context	ContextType	[DB]
userContext	UserContextType	[DB]
transient	EObject	[DB]
primitiveContext	PrimitiveContextType	[DB]
dynamicProperty	DynamicPropertyContextType	[DB]
propertySets	DynamicPropertySetType[] <DynamicProper...	[DB]
propertySets[0]	DynamicPropertySetType	[DB]
properties	PropertyType[] <PropertyType>	[DB]
properties[0]	PropertyType	[DB]
properties[1]	PropertyType	[DB]
name	String	[DB] AuditInputACS.enabled
value	String	[DB] false
type	String	[DB]
properties[2]	PropertyType	[DB]
properties[3]	PropertyType	[DB]
properties[4]	PropertyType	[DB]
name	String	[DB] createAccount_Callout.retryCount
value	String	[DB] 3
type	String	[DB]
properties[5]	PropertyType	[DB]
group	String	[DB] AccountCreationService_Med.ACS_Proxy
isPropagated	boolean	[DB]
correlation	EObject	[DB]
failInfo	FailInfoType	[DB]

Figure 9-40 Promoted properties loaded from the “US” mediation policy

In this case, the values for the retryCount and the AuditInput.enabled properties are different from those that we get in the previous test, according to the ACSMed_USPolicy that we created by using Business Space.

Looking at the Console view, shown in Figure 9-41, we can verify that this time there is no TraceMediation line, because the value of the Enabled property for the AuditInputACS primitive is false. This time we only can see the SystemOut lines, corresponding to the execution of the back-end service.



Task Flows Build Activities Properties Problems Server Logs Servers Console

WebSphere ESB Server v7.5 at localhost (WebSphere Application Server v7.0)

Timestamp	Level	Source	Message
4/18/11 15:36:33:473 EDT	ExportRuntime I	processMessage	
4/18/11 15:36:33:520 EDT	WSChannelFram A	CHF00019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:5	
4/18/11 15:36:34:379 EDT	TraceMediatio I	CWSXM3010I: Message 6A1D43E0-012F-4000-E000-041CC0A89188 audited at 4/18/11 3:36 PM	
4/18/11 15:36:36:613 EDT	WSChannelFram A	CHF00019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:5	
4/18/11 15:36:36:629 EDT	SystemOut	O Service: Account Creation Service	
4/18/11 15:36:36:629 EDT	SystemOut	O Version: 1.0	
4/18/11 15:36:36:629 EDT	SystemOut	O Account for 0815 created successfully!	
4/18/11 15:36:36:660 EDT	ExportRuntime I	processMessage	
4/18/11 15:44:28:707 EDT	ExportRuntime I	processMessage	
4/18/11 15:44:28:754 EDT	WSChannelFram A	CHF00019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:5	
4/18/11 15:44:30:113 EDT	WSChannelFram A	CHF00019I: The Transport Channel Service has started chain HttpsOutboundChain:localhost:5	
4/18/11 15:44:30:129 EDT	SystemOut	O Service: Account Creation Service	
4/18/11 15:44:30:129 EDT	SystemOut	O Version: 1.0	
4/18/11 15:44:30:129 EDT	SystemOut	O Account for 0815 created successfully!	
4/18/11 15:44:30:145 EDT	ExportRuntime I	processMessage	

Figure 9-41 Console view for the second test

At this point, we have extended and enhanced the mediation proxy module with advanced capabilities for retrieving service endpoints that are associated with SLAs. Also, we have implemented dynamic mediations using mediation policies.

Additional resources: For your convenience, we provide the following assets with partial and final solutions to the scenarios that we implemented in this chapter. You can find these assets in the `/Extend_Mediation/Results` folder of the Additional materials file.

- ▶ Result from the SLA Endpoint Lookup primitive part:
 - Named Queries, in the `/Part1 - SLAEndpointLookup/NamedQueries` sub-folder
 - Project Interchange, in the `/Part1 - SLAEndpointLookup/Project Interchange` sub-folder
- ▶ Result from the Policy Resolution Mediation primitive part:
 - Project Interchange, in the `/Part2 - PolicyResolution/Project Interchange` sub-folder
 - Mediation EAR file (for publishing in WSRR), in the `/Part2 - PolicyResolution/Mediation EAR file` sub-folder
 - WSRR export file (for importing in WSRR, containing the initial service and SLA entities and also the mediation module and the mediation policies), in the `/Part2 - PolicyResolution/WSRRExport` sub-folder



Service versioning

In SOA environments, multiple service versions can be running at the same time in a deployment environment. A service version in production can also exist in other non-production environments to facilitate development and testing meaning managing service versions is a challenge in SOA. This chapter describes how WebSphere Enterprise Service Bus Registry Edition (WESBRE) can help you manage service versions in your SOA.

10.1 Service versioning concepts

Managing versions of services is a key challenge in SOA. As a company's business processes change in response to new competitive challenges and business opportunities, so also services in an SOA will need to change to support the new processes.

One of the benefits of an SOA approach is that as it defines business functions as easily reusable services at a level of abstraction that separates the business function from the IT resources used to provide the function, the IT constraints on making changes to the business are reduced. This makes it easier to respond to changing business requirements by combining the existing services in new ways. It is easier to implement new products and processes, change existing ones, or recombine them to deliver new value, without requiring major changes to the underlying services.

However, inevitably the detailed functional and non-functional requirements for a business function will also change over time to meet the changing business requirements, and new versions of the service that implements a business function will be required.

To be successful in SOA it is essential to govern changes in versions effectively to minimize impact on continuous operations, to maximize reuse, and to meet the evolving business goals.

For example, the impact of introducing a new version needs to be understood. Does an existing version need to be kept in operation for a set period of time to ensure existing consumers have time to migrate to the new version, or can the change to the new version happen instantly?

Changes to a service may apply to a service specification, the actual code implementing the service, or any other meta-data related to a service, for example, a service level agreement (SLA) between a service consumer and provider. Because all of these components can change independently they can be versioned independently, even though they may belong to the same service overall.

In the sections that follow we describe the key service versioning concepts to follow when managing service versions with WESB Registry Edition.

10.1.1 Service versions

A *service* is defined by a combination of its *service interface* specification and its physical *implementation*. A service can change due to a change in the service

specification, a change in the service implementation, or changes in both. Service versions include the following types of changes:

- ▶ A change is *forward compatible* if the new version of the service can be deployed to coexist with the previous service versions, allowing consumers to continue consuming the existing versions using old SLAs. A deployment environment (for example, production) can have multiple forward-compatible versions of a service running simultaneously.
- ▶ A change is *backward compatible* if the new version is compatible with the previous versions in case the previous version is deprecated. The consumer of the previous version can consume the new version without any changes in the consumer request.

Changes to a service interface that are backward compatible might include the following examples:

- Adding a new operation (with no change in message type). If the consumer is unaware of the new operation, the old consumer request continues to work.
- Adding a new optional XML schema type that is not contained within the existing types.

Changes to service implementation that are backward compatible might include the following examples:

- Adding another QoS binding, for example JMS to HTTP
 - Updating the implementation code (software) to improve performance; that is, for better QoS
- ▶ A change is *non-backward compatible* if the existing consumer capability version can no longer consume the new service version. A new consumer request needs to be generated to conform to the new service interface or binding.

Changes that are non-backward compatible might include the following types of changes:

- Removing an operation
- Renaming an operation
- Changing structure of complex data type

Not all changes constitute a new service version. For example, a change in service bindings might not constitute a new service version. In some cases, a consumer version can use dynamic bindings. Thus, it continues to work uninterrupted by a binding change. For our discussion, we treat this type of change as a new service version because it often requires a change in the implementation of the provider service version and putting the change through deployment life cycle states.

10.1.2 WSDL document design and naming standards

Adopting naming standards in WSDL documents facilitates managing service versions and endpoints. Have naming standards that comply with the WSDL 1.1 specification, taking into consideration ease of operations and management.

Consider adopting the following standards in WSDL documents:

- ▶ Define service elements in a WSDL document that is separate from the portType and binding elements, to provide the most flexibility and ease of operations in governing service versions and endpoints.

Monolithic WSDL documents: There can be circumstances where services are already defined in monolithic WSDL documents. For information about how the governance enablement model allows management of a Web service defined in a monolithic document, refer to *Service Lifecycle Governance with IBM WebSphere Service Registry and Repository*, SG24-7793.

- ▶ Maintain a common *namespace* for all backward-compatible service versions to ensure backward compatibility of request messages between minor versions. You can include only a major version number in the target namespace of the service interface definition, as shown in Example 10-1.

Example 10-1 The namespace of the service interface element

```
targetNamespace="http://SupplyCompany.itso.ibm.com/AccountCreationV1"
```

- ▶ Adopt a common standard for naming the *namespace* of the Port and Service elements. This port or service namespace must be separate from the portType namespace when they are defined in a separate WSDL file from the portType element. For example, append */service* to the end of the service namespace, as shown in Example 10-2.

Example 10-2 The namespace for the /service element

```
targetNamespace="http://SupplyCompany.itso.ibm.com/AccountCreationV1  
/service"
```

- ▶ Include both major and minor version numbers in the *name* of the portType, binding, service, and port elements, as shown in Example 10-3.

Example 10-3 The portType name

```
<wsdl:portType name="AccountCreationV1_0">  
  <wsdl:operation name="createAccount">
```

```

        <wsdl:input message="intf:createAccount"
name="createAccount"/>
        <wsdl:output message="intf:createAccountResponse"
name="createAccountResponse"/>
    </wsdl:operation>
</wsdl:portType>

```

Example 10-4 shows the names of service and binding elements.

Example 10-4 The binding, service, and port names

```

<wsdl:binding name="AccountCreationV1_0_SoapBinding"
type="intf:AccountCreationV1_0">
    ..
    ..
</wsdl:binding>
<wsdl:service name="AccountCreationV1_0">
    <wsdl:port binding="serv:AccountCreationV1_0_SoapBinding"
name="AccountCreationServiceV1_0_Port">
        <wsdlsoap:address
location="https://supply.devhost:9443/AccountCreationV1_0/services/A
ccountCreationServiceV1_0_address"/>
    </wsdl:port>
</wsdl:service>

```

Version numbers: Although version number indicators is the preferred use, service versioning in WSRR does not require version number indicators in the name and namespaces of WSDL documents if you specify the version number property consistently when loading a WSDL.

For example, when you load a service WSDL document, the version property is used in combination with the name and namespace properties by the correlator modifier to either create a new *service* entity object in the service model, or to correlate it to the existing service entity.

See 6.2.1, “Service model” on page 145 for details about properties that define each service model entity and 6.7.1, “The correlator modifier” on page 188 for details about the correlator modifier.

- Adopt a common standard to name the *ports* of the same service version. Make the port names distinct and unique for each distinct QoS or binding.
- A service version can provide an SLD with endpoints in multiple deployment environments (test or production) or deployment architectures (gateway or proxy). In such cases, have the port names indicate each environment or

deployment architecture (gateway or proxy) to facilitate monitoring the observed service levels at the port level in various environments (for example, using ITCAM SOA) and to improve identification in WSRR.

Example 10-5 illustrates using another environment port for the same service version.

Example 10-5 Different environment port for the same service version

```
<wsdl:service name="AccountCreationV1_0">
  <wsdl:port binding="serv:AccountCreationV1_0_SoapBinding"
name="AccountCreationServiceV1_0_ProductionPort">
    <wsdlsoap:address
location="https://supply.prodhost:9447/AccountCreationV1_0/services/
AccountCreationServiceV1_0_url"/>
  </wsdl:port>
</wsdl:service>
```

For leading practices about defining schemas for versioning, refer to the following document:

<http://www.xfront.com/Versioning.pdf>

10.1.3 Service endpoint management

A *service endpoint* defines a location of the service implementation. A service version can be deployed at multiple service endpoints to facilitate deployment architecture, service availability, or load balancing. In an SOA environment, this location is often virtualized to service consumers. ESB architecture is often adopted to virtualize the service endpoints. The endpoints can be stored with other related service metadata in a service registry. You can have a mediation component in an ESB product look up service endpoints dynamically from the service registry.

The following typical changes for service endpoints of the same service version often occur:

- ▶ Change of service endpoints with the same QoS (for example, deploying an EAR file that contains implementation of a web service from a QA environment to a production environment)
- ▶ Change of service endpoint specific intent to change the QoS (SLD) (for example, deploying an EAR file that contains implementation of a web service to server clusters that provide various QoSs to provide new SLDs to consumers)

For the second type of change, where the intent is to change SLD, new SLAs might be required for consumer capability versions (including those with existing SLAs to old SLDs) to consume the new SLDs of the existing service version.

In the governance enablement profile SOA life cycle, the service version states include deployment states, such as staging and production. In SOA environments, a service is often consumed by other services, processes, or applications. As new consumers to a service version in production are developed, the production service version is made available in earlier environments such as staging. This availability allows testing of the new consumers' service without affecting the production environment.

A similar situation arises when adding endpoints to provide a new QoS for a service version already in production. Although there is no change in service implementation (code), the service version can be deployed to the new endpoints for testing in earlier staging environments. This deployment allows testing functions, such as endpoint lookup and verification of the new SLD. Meanwhile, the service version with existing endpoints (SLDs) continues to serve consumers of existing SLAs.

Deploying the service version: You can deploy a service version simultaneously in more than one environment. A service version in production (at the operational state of the SOA life cycle in the governance enablement profile) has at least a set of endpoints in production. Simultaneously, it can have endpoints with active SLAs that were deployed in earlier environments, such as staging.

10.2 Service versioning management

You can manage versioning of web services using several methods. Although leading practices are documented, there are no industry standards at the time of writing regarding service versioning. Many approaches depend on annotating version-related metadata in the WSDL documents that define a web service.

One method is to define a service version indicator in the namespace of elements of a WSDL document. An ESB can then extract the namespace from a consumer request and route it to the appropriate service version based on this data. When a service registry is used, another method is for an ESB to extract the version number from a consumer request and look up the service version based on the version number property stored in a service registry.

For details about service versioning, refer to the following documentation:

- *Best practices for Web services versioning*

<http://www.ibm.com/developerworks/webservices/library/ws-version/>

10.2.1 Managing service versions based on SLA

The governance enablement profile includes a service level agreement (SLA) to model contracts between the consumer and a service or capability version. WebSphere ESB can enforce the SLA and control access to services by checking whether a consumer has an SLA in place that allows the consumer to invoke the service before passing on the request to the service provider.

Similarly, as the SLA is linked to a specific service version rather than a business service, WebSphere ESB can use the SLA to control which version of a service is invoked for a specific consumer.

Before you continue: The service registry instance that we use in this chapter must contain the initial information that is required, such as WSDL documents and service metadata, as described in Chapter 7, “Registering services” on page 195.

For your convenience, we provide a WSRR export file (WSRRExport001.zip) that contains all the required definitions. You can find it in the /Basic_Mediation/StartingPoint folder of the additional material file.

To import this file into WSRR, launch the WSRR console, and in the Administrator perspective, select **Actions** → **Import** and import WSRRExport001.zip.

The SLA provides the technical-level agreement between a consumer and a provider of a capability version.

Figure 10-1 gives an overview of how an SLA between a consumer service version and a provider SLD is used to describe the service consumption model.

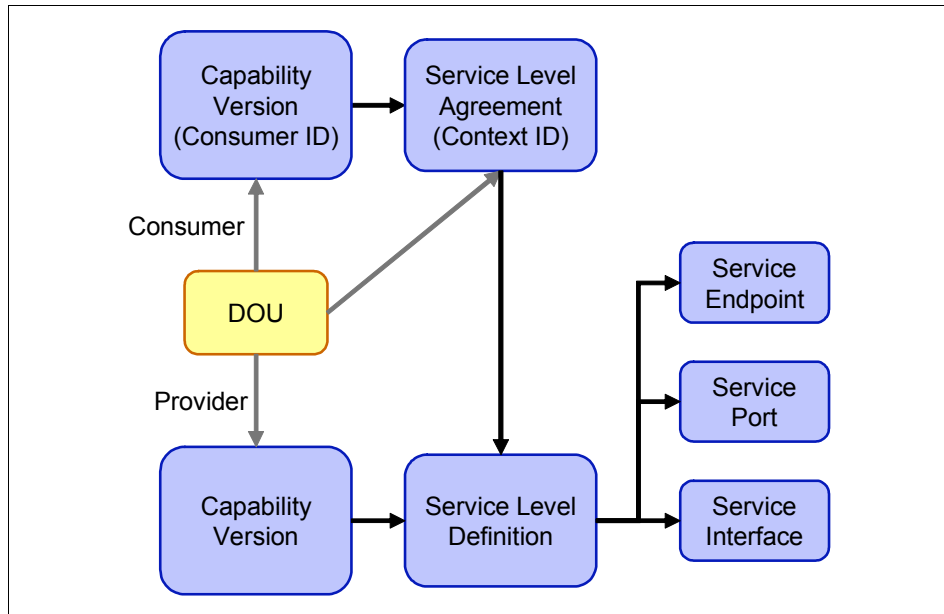


Figure 10-1 Service consumption model based on SLA

A document of understanding (DOU) is used to document a business-level agreement between a consumer capability version and a provider capability version. The DOU is the business view of the contract between provider and the consumer.

The consuming capability version has a consumer ID property that can uniquely identify the consumer. The SLA contains a context ID property to identify the appropriate SLA when a consumer has more than one SLA with the provider. A SLD can have more than one endpoint, for example an endpoint for each deployment environment.

Based on this relationship model, you manage service versions with the following requirements:

- ▶ A consumer must have an SLA with the provider SLD in WSRR to consume a provider capability version.
- ▶ Each consumer request for the capability must include a unique identifier for the consumer (for example, the consumer ID is added to the SOAP Header or elsewhere in the request).
- ▶ If the consumer has multiple SLAs, then have each consumer request also include an identifier that shows which SLA it intends to use (for example, the context ID is added to the SOAP Header or elsewhere in the request).
- ▶ An ESB can then enforce these requirements and route consumer requests based on the consumer ID and context ID to the appropriate endpoints of the correct service version.

WebSphere ESB provides a number of mediation primitives to interact with WSRR. For details about these primitives refer to Chapter 8, “Implementing a mediation” on page 251 and Chapter 9, “Extending the mediation” on page 315.

In the following example we use the SLA Endpoint Lookup primitive within a dynamic service gateway pattern mediation to manage service versions.

Figure 10-2 on page 363 illustrates how the SLA endpoint lookup primitive works.

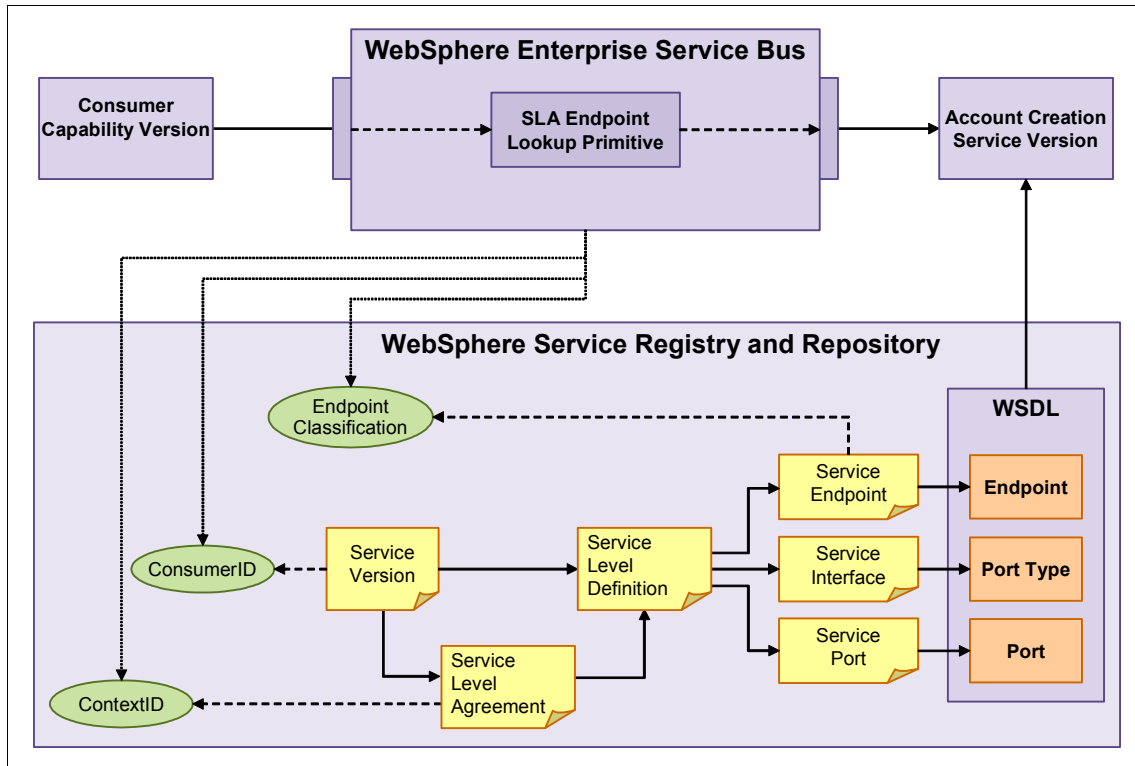


Figure 10-2 SLA endpoint lookup mediation primitive in WebSphere ESB

The SLA Endpoint Lookup primitive queries WSRR for a consumer service version with the matching consumer ID and the related active SLA agreement with the matching context ID.

Using the related SLDs, the query returns all the endpoints listed. The query can be modified to return only endpoints with specific classifications, for example those that are online in production.

Note: WebSphere ESB provides caching of the SLA endpoint lookup queries for improved performance.

10.2.2 Support service versioning with a dynamic SLA gateway solution

In this section, we introduce the dynamic SLA gateway solution that we use in this chapter and explain how to prepare it for service versioning.

The dynamic service gateway pattern in WebSphere ESB accepts generic anyType messages, meaning it is able to process request messages for any service and then determine the endpoint address of the service implementation that is to be invoked within a mediation.

In our solution, the gateway inspects the SOAP message header of each service request and extracts the consumer ID and the SLA context ID from the governance enablement profile gateway header. It uses these to determine what version of a service to invoke.

Importing the solution into IBM Integration Designer

To import the dynamic solution into IBM Integration Designer, follow these steps:

1. Launch Integration Designer, and specify a new workspace. Then, in Integration Designer, click **File** → **Import**.
2. In the Import dialog box, expand **Other**, select **Project Interchange**, and click **Next**. Click **Browse** to locate DynamicSLAGW_Solution.zip in the additional material that is provided with this book. Click **Open**, and then click **Finish** to import the project.

Additional material: You can find all additional material resources for this chapter in the /Versioning folder of the additional material that is supplied with this book.

The DynamicSLA_Gateway module is imported into your workspace. This module contains a simple dynamic gateway pattern, which we have customized for this example.

To examine the DynamicSLA_Gateway module, in the Business Integration view, expand **DynamicSLA_Gateway** and double-click **Assembly Diagram**. The Assembly Diagram shows three components, as shown in Figure 10-3:

- A SOAP web services export
- A mediation flow component
- A SOAP web services import

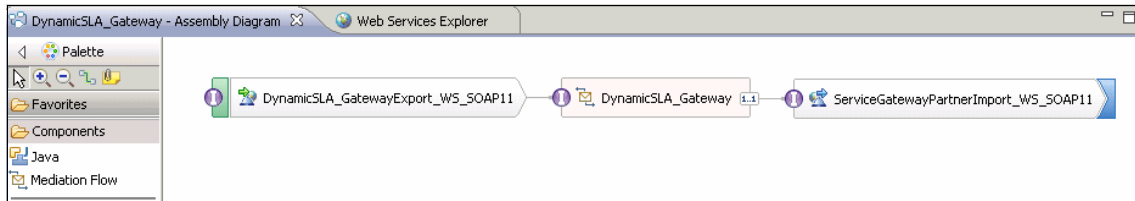


Figure 10-3 Assembly for the dynamic SLA gateway module

3. In the Business Integration view, double-click **Dependencies**. Expand **Predefined Resources**. Note that we selected **Governance Enablement Profile Gateway Header** for use in the module, as shown in Figure 10-4. The header schema includes the consumer ID and context ID elements.

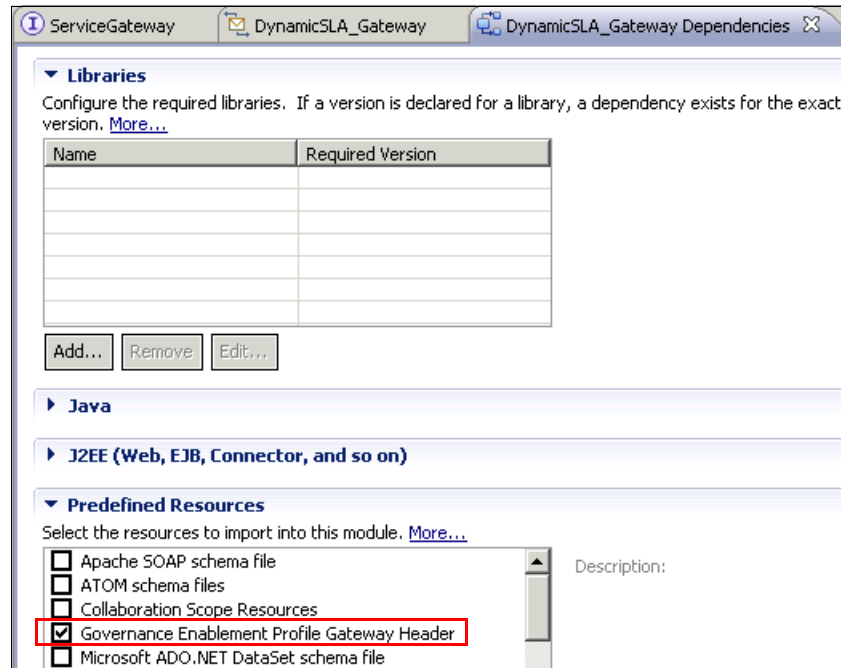


Figure 10-4 Using the predefined Governance Enablement Profile Gateway Header

4. In the Business Integration view, select **Integration Logic** → **Mediation Flows**, and double-click **DynamicSLA_Gateway**. The Operation connections view shows two mediation flows: one that is defined for the requestOnly operation, and one that is defined for the requestResponse operation, as shown in Figure 10-5.

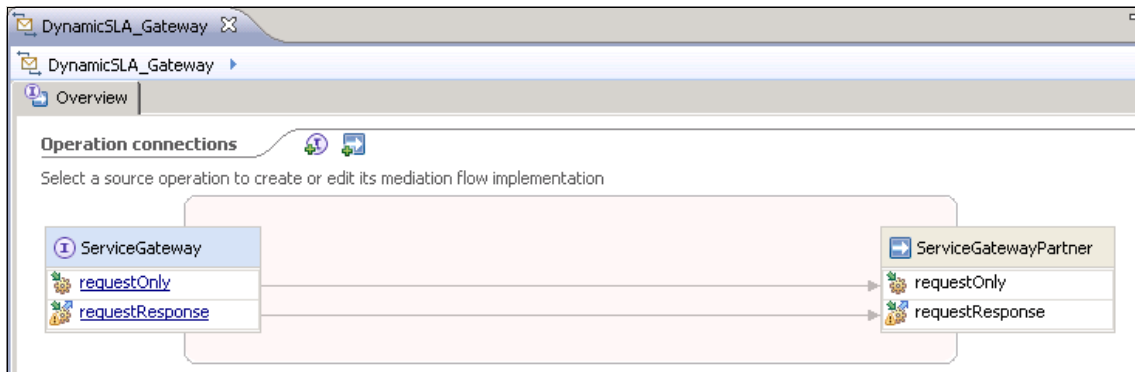


Figure 10-5 Operation connections of DynamicSLA_Gateway

5. Click the **requestResponse** operation to view this mediation flow. Figure 10-6 on page 366 shows the request flow for this mediation. We have customized this flow to contain the following mediation primitives:
 - Trace mediation primitives to trace elements to the local server log
 - An SLA Endpoint Lookup mediation primitive
 - A fail mediation primitive

In this mediation flow, a service provider is invoked if the SLA Endpoint Lookup primitive returns a match. If there is no match, the flow ends with a failure. The trace primitives are used for debugging and can be turned on and off using promoted properties.

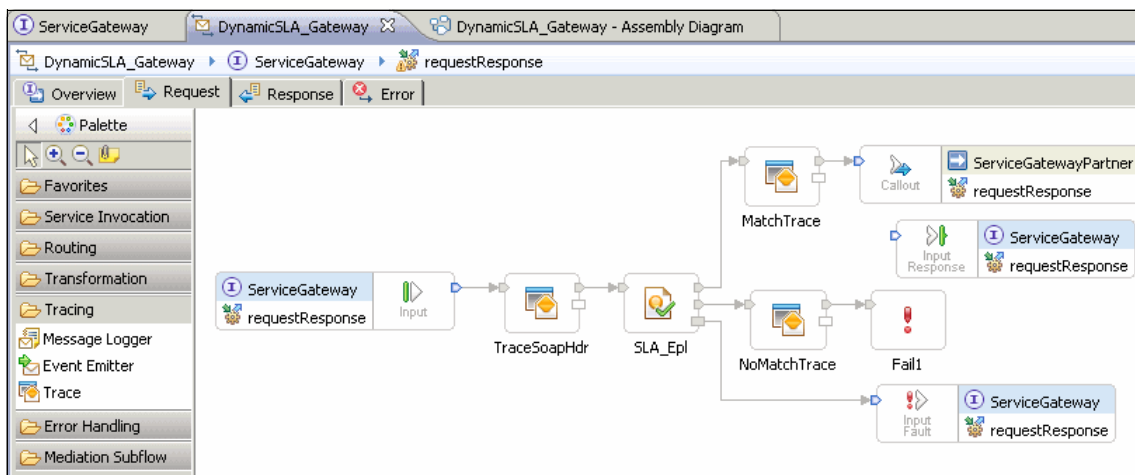


Figure 10-6 Request message flow

6. Select the **SLA_Epl** primitive, and select the Details tab in the Properties view. The default property settings displays, as shown in Figure 10-7.

The primitive will use the values set in the ConsumerID and ContextID fields of the Governance Enablement Profile Gateway Header in the request message to query the registry for a valid SLA. It will return matching Endpoints that are classified as being in the development environment.

Figure 10-7 Detail properties of the SLA endpoint lookup primitive

7. Still in the Properties view, click **Promotable Properties** (Figure 10-8). The Registry Name and Endpoint Classification properties are specified as promoted properties, which allows you to change these properties to appropriate values in the Integrated Solutions Console.

Property	Promoted	Group	Alias	Alias value
Registry Name	<input checked="" type="checkbox"/>	DynamicSLA_Gatewa...	SLA_Epl.registryName	WSRRDefault
Consumer ID	<input type="checkbox"/>			
Context ID	<input type="checkbox"/>			
Endpoint Classificat...	<input checked="" type="checkbox"/>	DynamicSLA_Gatewa...	SLA_Epl.environment...	http://www.ibm.com/...

Figure 10-8 Promotable properties for the SLA Endpoint Lookup mediation primitive

10.2.3 Publishing the gateway endpoint for each service version

In this scenario, services are accessed by all consumers through the dynamic SLA gateway.

Therefore, the gateway endpoint needs to be added to the WSDL for each service version invoked by the gateway and published in WSRR so that consumers invoke the gateway endpoint to access the services rather than accessing the service directly.

10.3 Service versioning scenarios

In this section, we demonstrate how the dynamic SLA gateway solution handles the following scenarios for service version management:

- ▶ Manage requests to an existing service version
- ▶ Manage making a backward-compatible change that does not change to an existing service interface
- ▶ Manage making a backward-compatible change to a service interface
- ▶ Manage making a change to a service interface that is not backward compatible

The dynamic SLA gateway solution can support all of these scenarios with no additional coding or change in the mediations.

Version numbers: We adopt the service version number pattern of $V_{x.y}$, as explained here:

- ▶ The first digit, x , indicates a major version for non-backward compatible changes.
- ▶ The second digit, y , represents backward-compatible changes or a minor version number.

We use $V_{x.y.z}$ where the third digit, z , represents a subminor version with the same service interface.

10.3.1 Processing requests for version 1.0 of the service

First we show consuming the initial 1.0 version of a service using the dynamic SLA gateway solution.

Figure 10-2 shows how the SLA endpoint lookup mediation primitive assigns an endpoint based on the SLA between a consumer capability version 1.0 and the version 1.0 SLD of this provider service.

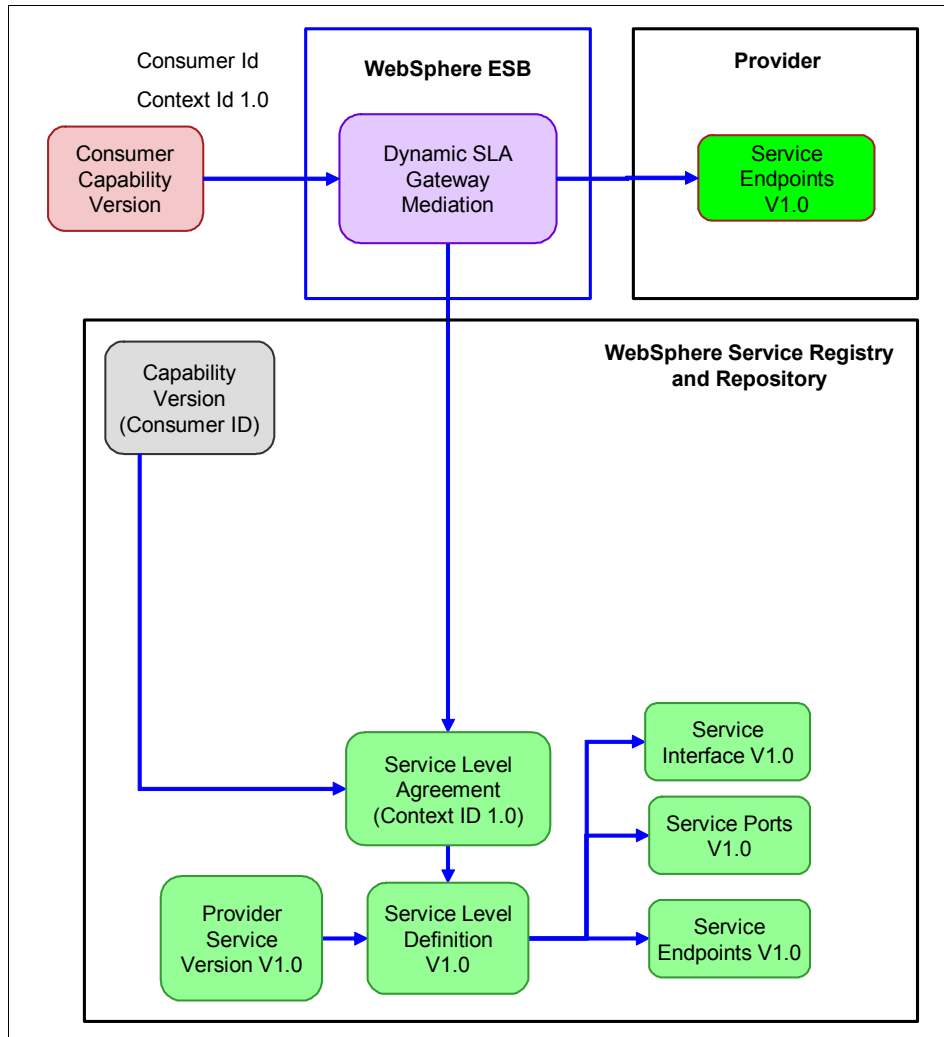


Figure 10-9 Service endpoint V1 lookup based on SLA

Configuring the v1.0 provider

In 7.4, “Registering a service using Business Space” on page 211, we illustrate how to register the initial version of account creation service in WSRR and how it is governed through the appropriate life cycles.

Follow the steps that are described in 7.4, “Registering a service using Business Space” on page 211 through to 7.4.5, “Providing a specification and service level definition for a service” on page 232.

After completing those steps the account creation service v1.0 is at the *Realized* state in the SOA life cycle. The development team has implemented the service provider and has deployed the EAR file for the account creation service v1.0 (named AccountCreationV1_0EAR.ear) to a server in development environment. The team has also performed functional tests (without the ESB gateway solution.)

Publishing the gateway provider endpoint

Optionally, you can register the gateway endpoint for the account creation service v1.0 that is made visible to consumer capability versions. Follow these steps:

1. Load the gateway endpoint WSDL for the account creation service v1.0 AccountCreationV1_0_DevelopmentPortGateway.wsdl into WSRR.
2. Add the staging gateway port to the SLD.
3. Add the staging gateway service endpoint to the SLD.
4. Authorize the gateway endpoint for use to change its state to online.

Configuring the v1.0 consumer

To configure a consumer capability in WSRR, complete the steps described in 7.4.6, “Governing the service consumer” on page 238.

By following these steps you now have the consumer application version 1.0 in the *Specified* state of the SOA life cycle environment. The consuming application is able to consume the account creation service through the dynamic SLA gateway.

We use the Web Services Explorer in IBM Integration Designer to send requests to the provider service, which simulates the consumer application.

Testing

To test the SLA gateway solution for managing the service version 1.0 in the development environment, complete the following steps:

1. In IBM Integration Designer, open the context menu by right-clicking a WSDL in any web services project.
2. In the context menu, select **Web services** → **Test with Web Services Explorer**.

Web Services Explorer: Using the Web Services Explorer in IBM Integration Designer allows us to test the versioning as part of the final verification steps in the *Realized* state in the development environment. In many cases, such tests involve multiple consumer and provider services. Thus, you can complete such tests only when all involved services are deployed in a staging environment. Test version management early, when possible, according to your test plan.

3. Clear the WSDL entry under the WSDL main in the navigator view by clicking the eraser icon, as shown in Figure 10-10.

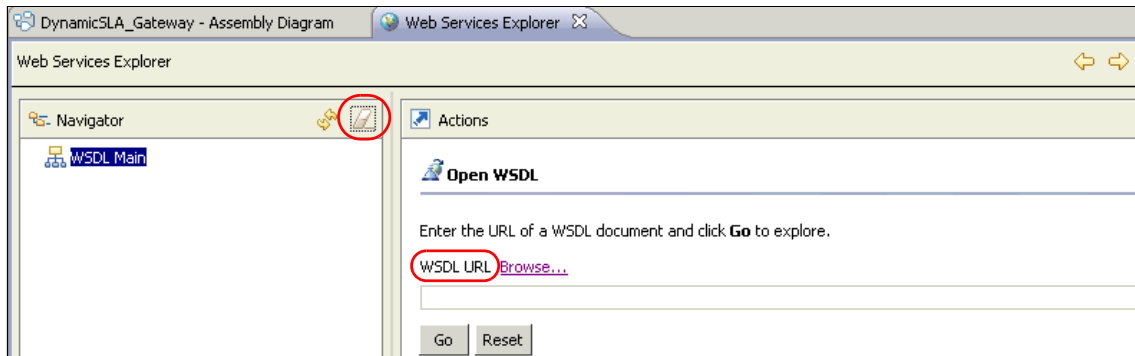


Figure 10-10 Clearing the WSDLs in the Web Services Explorer

4. In the WSDL URL field, enter the following web services URL for the account creation to bring the WSDL into the navigator view:

`http://localhost:9080/AccountCreationV1_0/services/AccountCreationServiceV1_0_DevelopmentPort?wsdl`

Figure 10-11 shows the completed WSDL URL field.

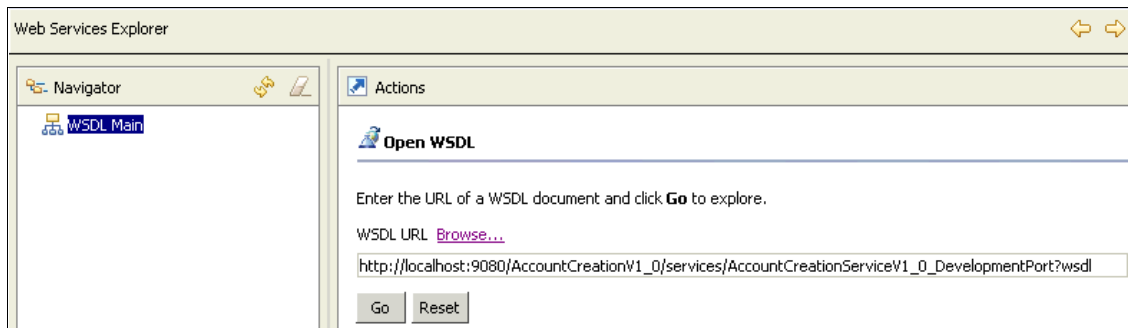


Figure 10-11 Fetching the WSDL into Web Services Explorer

- Click **Go** to bring the WSDL into the navigator view as shown in Figure 10-12. Select the endpoint, and click **Add**.

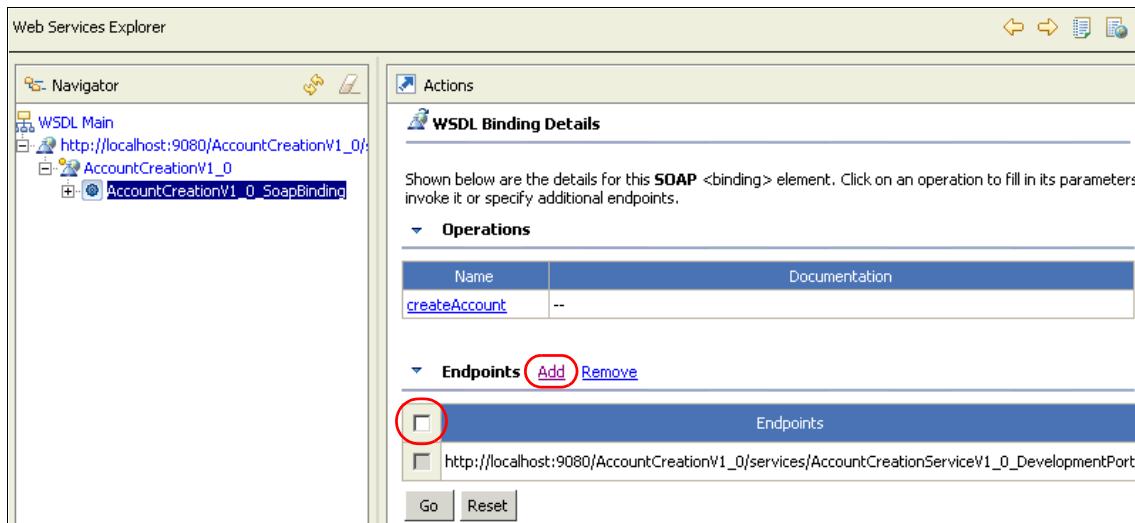


Figure 10-12 Details of the WSDL binding in Web Services Explorer

- In the new second entry under Endpoints, copy and paste the service endpoint for the dynamic SLA gateway, as follows:
`http://localhost:9080/DynamicSLA_GatewayWeb/sca/DynamicSLA_GatewayExport_WS_SOAP11`

Note: In our test cases, both the WSRR server and WebSphere ESB server are on localhost in a development topology.

- Select this entry and click **Go**, as shown in Figure 10-13, to set the endpoint to the gateway endpoint.

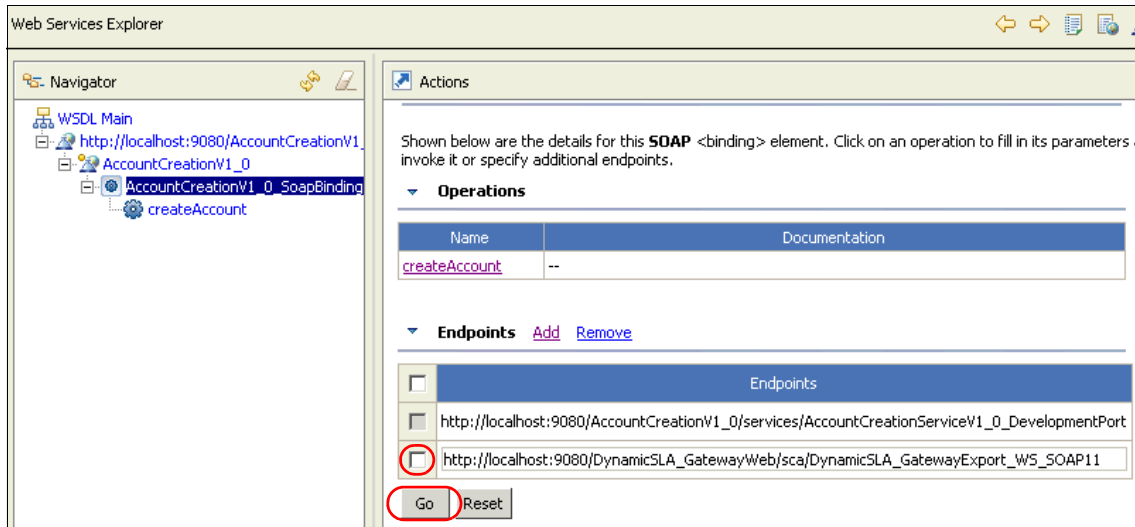


Figure 10-13 Changing the endpoint to the dynamic SLA gateway endpoint address

8. Now click **createAccount** to open the UI to enter the input data. Enter input data as shown in Figure 10-14.

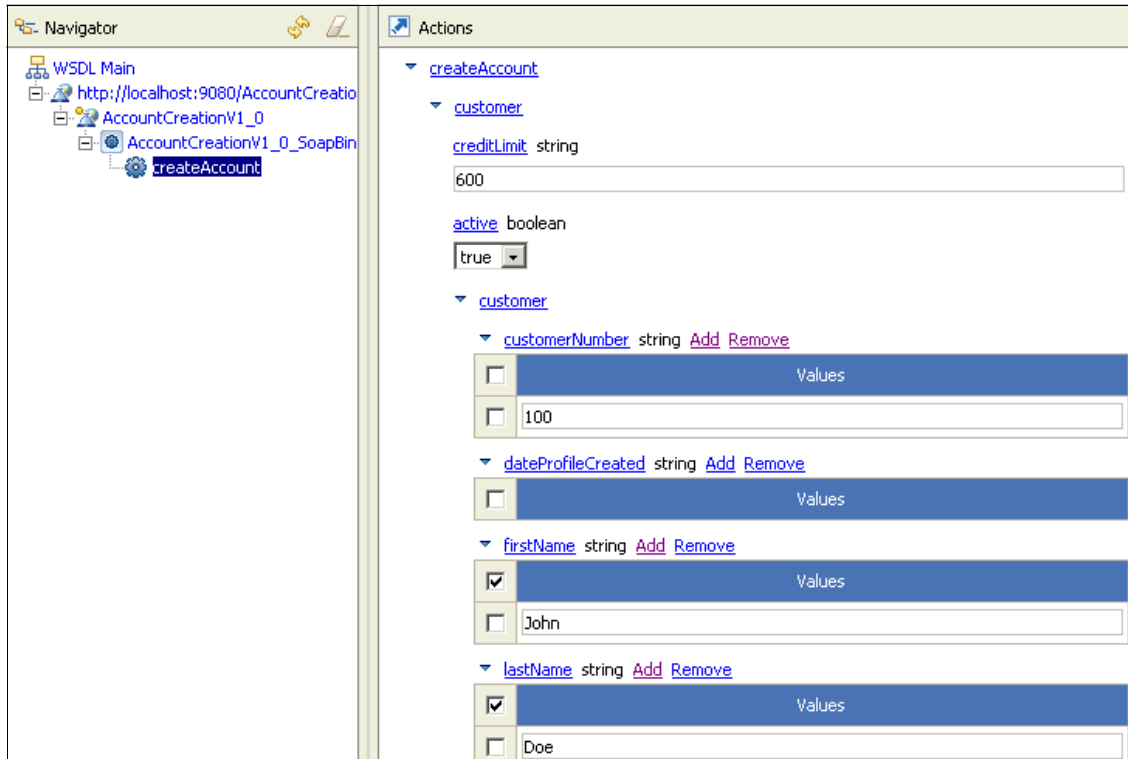


Figure 10-14 Entering input data in the Web Services Explorer

- Click the source link at the top corner to open the view for the request header, as shown in Figure 10-15. Enter the following XML text for the governance enablement profile gateway header:

```
<p:GEPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema
GEPGatewayHeader.xsd ">
  <consumerID>AMC100</consumerID>
  <contextID>AMC100CI01</contextID>
</p:GEPGatewayHeader>
```

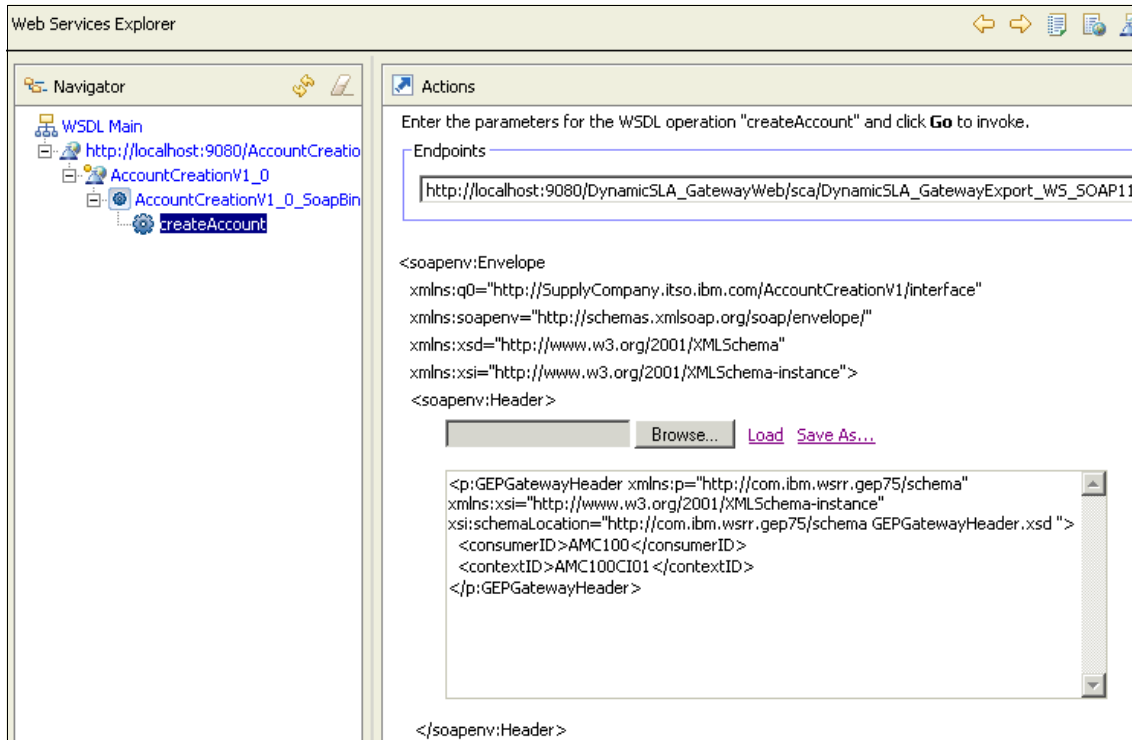



Figure 10-15 Entering the governance enablement profile gateway header data in the Web Services Explorer

The overall SOAP request, including the header and body, looks similar to the image shown in Figure 10-16.



Figure 10-16 The header and body of the SOAP message in the source view

10. Click **Go** to send the request for the account creation service to the dynamic SLA gateway.
11. Verify the response message, as shown in Figure 10-17.

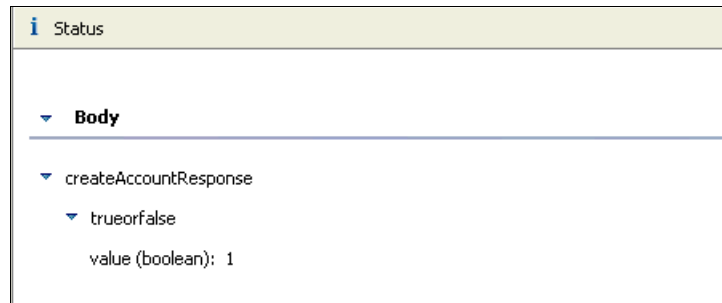


Figure 10-17 The response message from the account creation service V1.0 in the Web Services Explorer

12. Check that the console view shows the output log of the WebSphere ESB Registry Edition server. The log displays the message shown in Figure 10-18. This message illustrates that the dynamic SLA gateway routes the request to the correct service version endpoint.

```
[4/13/11 2:48:41:862 EDT] 0000009f SystemOut      O Account for 100 created successfully !!!
```

Figure 10-18 The message logged by the account creation service V1.0

10.3.2 Introducing a new sub-minor service version

A sub-minor change does not change the service interface but does change another aspect of the service. Following our naming convention, this is a change from v1.0.0 to version 1.0.1.

An example of this is demonstrated in the following scenario where the SLD for the account creation service is updated because quality of service (QoS) requirements have changed, thus requiring the account creation service response time to be reduced and for the service to have increased availability.

The QoS improvements are achieved by deploying the account creation service on high-end servers with higher availability with no change to the service interface.

Existing consumers do not need their implementation of the service call to be updated to use the new version. All they have to do to use the new service version is to put in place a new SLA to take advantage of the changes.

Consumers who require the improved QoS will create a new SLA to use the service. Existing consumers who do not require the improved QoS will continue to use their existing SLAs without requiring any changes to be made.

The service gateway supports this scenario by routing requests from consumers who have created a new SLA with the new service version to the updated version, and routing requests from consumers that have not updated their SLA to the original version of the service.

Figure 10-19 illustrates how the SLA endpoint lookup primitive helps select the version 1.0.1 of the service for existing versions of the consumer application based on SLAs.

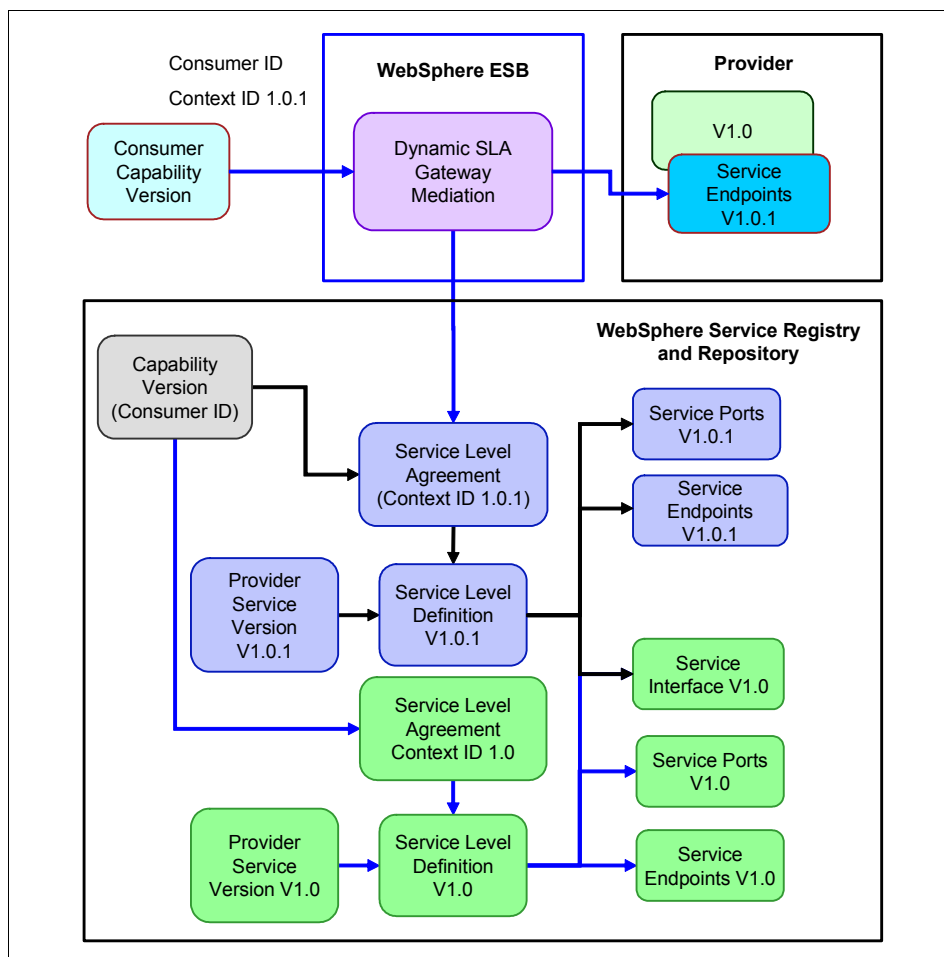


Figure 10-19 Service version management creating a new service version 1.0.1

Configure the v1.0.1 provider

1. Follow the steps described in 7.4.3, “Registering a service” on page 220. Note that you do not need to load an interface WSDL because version 1.0.1 uses the same service interface as Version 1.0 so you only need to load the endpoint WSDL named `AccountCreationV1_0_1_DevelopmentPort.wsdl`.
2. Then, follow the steps described in 7.4.4, “Providing scope and planning information for a service” on page 229 and 7.4.5, “Providing a specification and service level definition for a service” on page 232.
3. At this point, the account creation service V1.0.1 is at the *Realized* state. Deploy the EAR file for the account creation service V1.0.1 (named `AccountCreationV1_0_1EAR.ear`) to a server in the development environment.

4. Classify the development service endpoint for V1.0.1 on test server with the environment classification *Development*.

Next, we configure the consumer application to consume the consumer service V1.0.1.

Configure the existing consumer service v1.0

In this scenario, the existing consumer application is already in the *Operational* state in the production environment. The consumer application version can consume the new service V1.0.1 without any change.

To configure the consumer application to consume the existing consumer service v1.0, establish an SLA for the consumer application v1.0 to consume the provider account creation service V1.0.1 following the steps described in 7.4.6, “Governing the service consumer” on page 238. Specify the context ID property for the SLA to be AMC100CI02. Figure 10-20 shows the resulting SLA details.

The screenshot shows a web application window titled "Service Registry Detail". It contains a table with the following sections:

Service Registry Detail	
Action	
AMC100 Account Creation Service (1.0.1) SLA	
Service Level Agreement Properties	
Name:	AMC100 Account Creation Service (1.0.1) SLA
Description:	
Context Identifier:	AMC100CI02
Subscription	Thursday, April 21, 2011
Availability Date:	
Subscription	Thursday, April 21, 2011
Termination Date:	
Version Match	LatestCompatibleVersion
Criteria:	
Governance State	
Governance State: SLA Active	
Relationships	
Agreed Endpoints	
Name	Governance State
SLD - Account Creation Service (1.0.1)	SLD Subscribable
Consuming Capability Version	
Name	
Account Management Consumer (1.0.0)	
Account Management Consumer	

Figure 10-20 Active SLA for the account management consumer v1.0 to consume the account creation service v1.0.1

Testing

For our tests, we use the Web Services Explorer in IBM Integration Designer to simulate the consumer to send requests to the account creation service through the dynamic SLA gateway.

To test the SLA gateway solution for managing the service version 1.0.1 in the development environment, follow these steps:

1. In IBM Integration Designer, open the context menu by right-clicking a WSDL in any web services project.
2. In the context menu, select **Web services** → **Test with Web Services Explorer**.

3. Clear the WSDL entry under WSDL main in the navigator view by clicking the eraser icon.

4. In the WSDL URL field, enter the following web services URL for the account creation to bring the WSDL into the navigator view:

```
http://localhost:9080/AccountCreationV1_0_1/services/AccountCreationServiceV1_0_1_DevelopmentPort?wsdl
```

5. Click **Go** to bring the WSDL into the navigator view. Observe that ESB mediations route the new consumer request to the correct provider service version and endpoints per the SLA.

6. Select the endpoint, and click **Add**.

7. In the new second entry under Endpoints, copy and paste the service endpoint for the dynamic SLA gateway:

```
http://localhost:9080/DynamicSLA_GatewayWeb/sca/DynamicSLA_GatewayExport_WS_SOAP11
```

8. Select this entry, and click **Go** to set the endpoint to the gateway endpoint.

9. Next, open the source view for the input data, and enter the following XML text for the governance enablement profile gateway header:

```
<p:GEPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema
GEPGatewayHeader.xsd ">
  <consumerID>AMC100</consumerID>
  <contextID>AMC100CI02</contextID>
</p:GEPGatewayHeader>
```

10. Click **Go** to send the request for the account creation service to the dynamic SLA gateway.

The response message will look as shown in Figure 10-21.

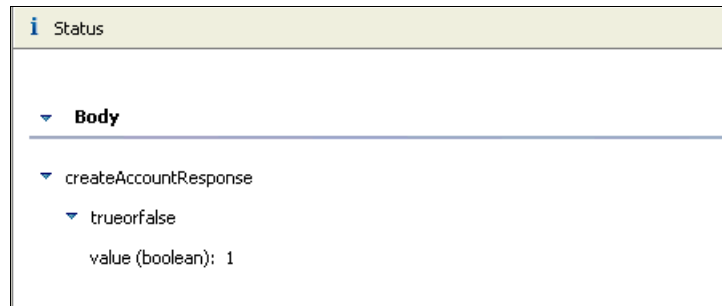


Figure 10-21 The response message from the account creation service V1.0.1 in the Web Services Explorer

Verify that the console view shows the output log of the WebSphere ESB Registry Edition server. The log displays the message shown in Figure 10-22. This message indicates that the dynamic SLA gateway routes the request to the correct service version 1.0.1 endpoint.

```
[4/15/11 1:06:10:174 EDT] 0000007d SystemOut      O Service: Account Creation Service
[4/15/11 1:06:10:174 EDT] 0000007d SystemOut      O Version: 1.0.1
[4/15/11 1:06:10:174 EDT] 0000007d SystemOut      O Account for 200 created successfully!
```

Figure 10-22 The log message from account creation service version 1.0.1

Retiring the provider service version 1.0

The provider version 1.0 is put into the *Superseded* state to indicate that it is about to be deprecated. Existing consumers of service version 1.0 are notified and given time to migrate to the newer version during the *Superseded* state and before the *Deprecated* state. Thus, an existing consumer of V1.0 needs to register a new SLA against the provider V1.0.1 in WSRR. In the interim period, you can allow consumer capability V1.0 to consume either provider version 1.0 or version 1.0.1, based on the context ID in its request header.

As the service version 1.0 is retired, all SLAs that consume its SLDs are deactivated into the *Inactive* state. These SLAs are then terminated. The consumer requests that include the context ID of the inactive or terminated SLA are rejected by the SLA gateway solution. Because the change from service version 1.0 to 1.0.1 is backward compatible, the existing consumer (for example, the account management consumer 1.0) can send the same consumer request to the provider V1.0.1. You need to update only its governance enablement profile message header with the correct context ID for the new SLA that is registered in WSRR.

10.3.3 Introducing a new a minor service version

A minor version change makes a backward-compatible change to a service interface, for example by adding a new operation. A minor version change is characterized by the fact that existing consumers can interact with the updated service provider unchanged as long as they do not need to use any of the new functionality. Hence, any minor version update has no impact on existing consumers at all, and they can either be left alone completely, or be migrated to take advantage of potential new features in the service over time. Following our naming convention, this is a change from v1.0.0 to version 1.1.0.

In the following scenario we want to extend the account management portal to verify that an account was created given consumer information. Analysis of existing interfaces indicate that the best method to use is to add a new operation to the existing account creation service. This addition is a backward-compatible change of the account creation service interface.

Figure 10-23 illustrates how the SLA endpoint lookup primitive routes the request from the existing consumer version to the appropriate service version based on the new SLA.

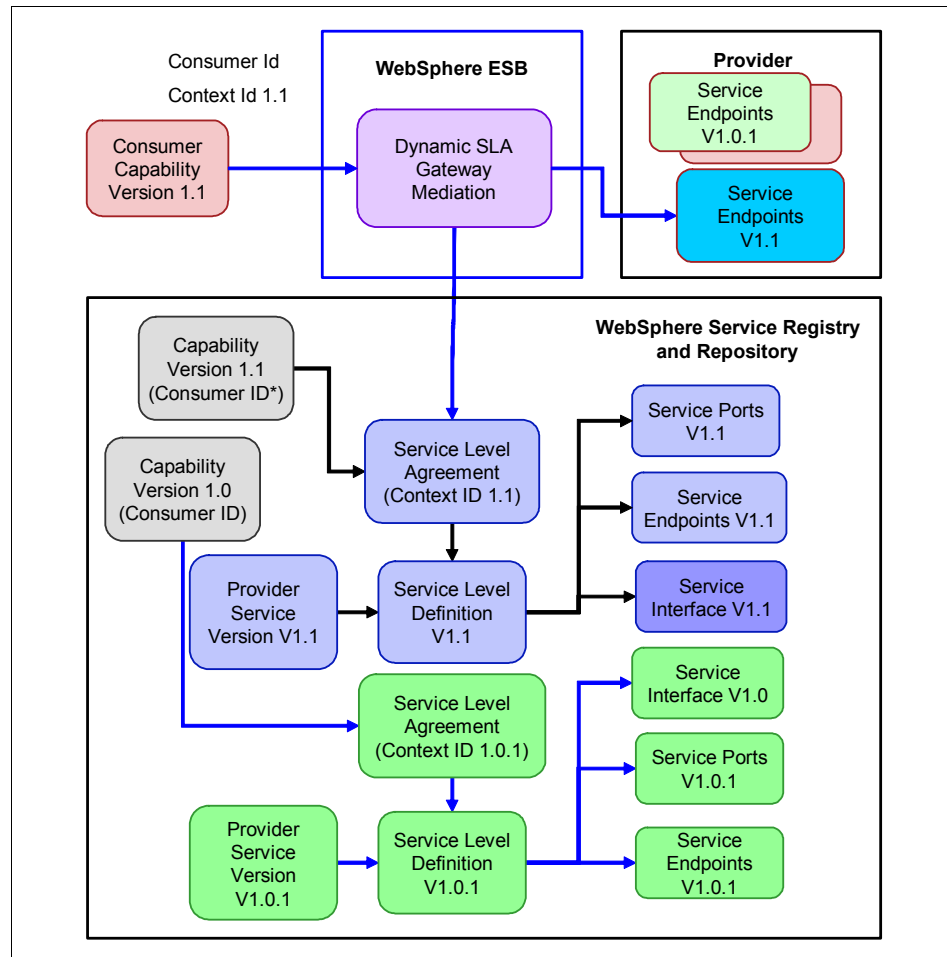


Figure 10-23 Managing the new account creation minor service version 1.1

Configuring the v1.1 provider

To configure the service provider in WSRR, complete these tasks:

1. Follow the steps described in 7.4.3, “Registering a service” on page 220. Note that you do not need to load an interface WSDL because version 1.0.1 uses the same service interface as version 1.0. You only need to load the endpoint WSDL named `AccountCreationV1_1_DevelopmentPort.wsdl`.

2. Then, follow the steps described in 7.4.4, “Providing scope and planning information for a service” on page 229 and 7.4.5, “Providing a specification and service level definition for a service” on page 232.
3. At this point, the account creation service V1.1 is at the *Realized* state. Deploy the EAR file for the account creation service V1.1 (named AccountCreationV1_1EAR.ear) to a server in the development environment.
4. Classify the development service endpoint for V1.1 on a test server with the environment classification *Development*.

Next, we configure the consumer application to consume the consumer service V1.1.

Configuring the v1.1 consumer

You can configure the consumer service V1.1 to govern a new consumer application version 1.1 that consumes the account creation service v1.1 following the steps as described in 7.4.6, “Governing the service consumer” on page 238 with the following changes:

1. You do not need to identify a requirement for a new business capability because you use the existing business service. Thus, you have two versions (1.0.0 and 1.1.0) of the account management consumer under the account management portal capability.
2. Specify a consumer ID property of AMC110 for this application version. For the SLA, specify its context ID property to be AMC110CI01.

At this point, the consumer application version 1.1 in the *Specified* state of the SOA life cycle to begin development of its implementation. It is now ready to send consumer requests to the account creation service using the dynamic SLA gateway in the test environment.

Testing

To test the SLA gateway solution for managing the service version 1.1 in the development environment, complete the following steps:

1. In IBM Integration Designer, open the context menu by right-clicking a WSDL in any web services project.
2. In the context menu, select **Web services** → **Test with Web Services Explorer**.
3. Clear the WSDL entry under WSDL main in the navigator view by clicking the eraser icon.
4. In the WSDL URL field, enter the following web services URL for the account creation to bring the WSDL into the navigator view:

`http://localhost:9080/AccountCreationV1_1/services/AccountCreationServiceV1_1_DevelopmentPort?wsdl`

5. Click **Go** to bring the WSDL into the navigator view as shown in Figure 10-24.

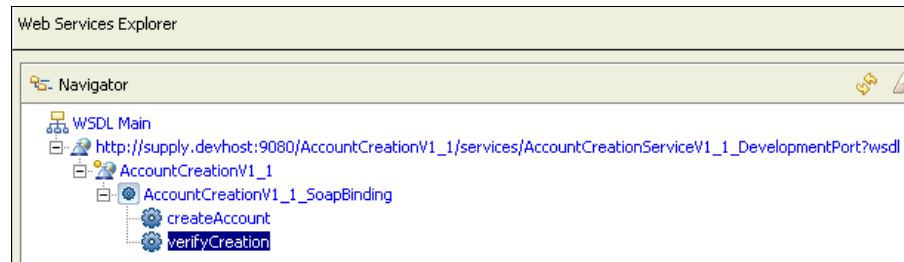


Figure 10-24 The WSDL for the account creation service v1.1 in the WSDL navigator

6. Select the endpoint, and click **Add**.
7. In the new second entry under Endpoints, copy and paste the service endpoint for the dynamic SLA gateway:
`http://localhost:9080/DynamicSLA_GatewayWeb/sca/DynamicSLA_GatewayExport_WS_SOAP11`

8. Select this entry, and click **Go** to set the endpoint to the gateway endpoint.
9. Open the source view for the input data for the createAccount operation. In the SOAP header window, enter the following XML text for the governance enablement profile gateway header:

```
<p:GEPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema
GEPGatewayHeader.xsd ">
  <consumerID>AMC110</consumerID>
  <contextID>AMC110CI01</contextID>
</p:GEPGatewayHeader>
```

10. Enter the following XML text for the body:

```
<q0:createAccount>
  <customer>
    <creditLimit>550</creditLimit>
    <active>true</active>
  </customer>
  <customer>
    <customerNumber>200</customerNumber>
    <firstName>Nay</firstName>
    <lastName>Boss</lastName>
  </customer>
</createAccount>
```

</q0:createAccount>

11. Click **Go** to send the request for the account creation service to the dynamic SLA gateway.
12. Verify the response message as shown in Figure 10-25.

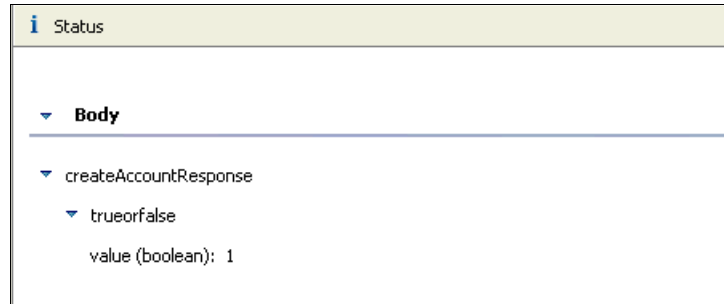


Figure 10-25 The response message from the account creation service V1.0.1 in the Web Services Explorer

Verify that the console view shows the output log of the WebSphere ESB Registry Edition server. The log displays a message as shown in Figure 10-26.

```
[5/13/11 18:50:52:127 EDT] 000000ce SystemOut      O Service: Account Creation Service
[5/13/11 18:50:52:127 EDT] 000000ce SystemOut      O Version: 1.1
[5/13/11 18:50:52:127 EDT] 000000ce SystemOut      O Account for 200 created successfully!
```

Figure 10-26 The log message from account creation service version 1.1

13. You can also test the new operation that is added to the service interface V1.1, verifyAccount. In the source view of the input for the verifyAccount operation, enter the following header XML text:

```
<p:GEPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema
  GEPGatewayHeader.xsd">
  <consumerID>AMC110</consumerID>
  <contextID>AMC110CI01</contextID>
</p:GEPGatewayHeader>
```

14. In the source view of the input for verifyAccount operation, enter the following XML text:

```
<q0:verifyCreation>
  <customer>200</customer>
</q0:verifyCreation>
```

The resulting input data in source view looks similar to that shown in Figure 10-27.



```
<soapenv:Header>
  <p:GEPPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema GEPPGatewayHeader.xsd">
    <consumerID>AMC110</consumerID>
    <contextID>AMC110CI02</contextID>
  </p:GEPPGatewayHeader>
</soapenv:Header>
<soapenv:Body>
  <q0:verifyCreation>
    <customer>300</customer>
  </q0:verifyCreation>
</soapenv:Body>
```

Figure 10-27 Input XML data for the verifyAccount operation

15. Click **Go**. A success message displays in the output log console, as shown in Figure 10-28. This message indicates that the dynamic SLA gateway routes the request to the correct service version 1.1 endpoint.

```
[5/13/11 20:59:24:627 EDT] 000000fe SystemOut    O Service: Account Creation Service
[5/13/11 20:59:24:627 EDT] 000000fe SystemOut    O Version: 1.1
[5/13/11 20:59:24:627 EDT] 000000fe SystemOut    O AccountCreation for 300 verified successfully
```

Figure 10-28 The output log message for verifyAccount operation of account creation service version 1.1

10.3.4 Introducing a new major service version

A new major service version changes the service interface in such a way that it is not backwardly compatible, so consumers must also change if they want to use the new service. Following our naming convention, this is a change from v1.0.0 to version 2.0.0.

For this scenario, financial regulations require additional customer information for opening accounts. The analysis shows that the modifications that are required cannot be done to the existing input data structure without breaking the existing consumers. Thus, the change in the interface is not backward compatible. You need to create a new major version of the account creation service.

A new SLA is established between the account management application version 2.0 and the account creation service version 2.0. Figure 10-29 shows how the SLA endpoint lookup mediation finds a major version of the service.

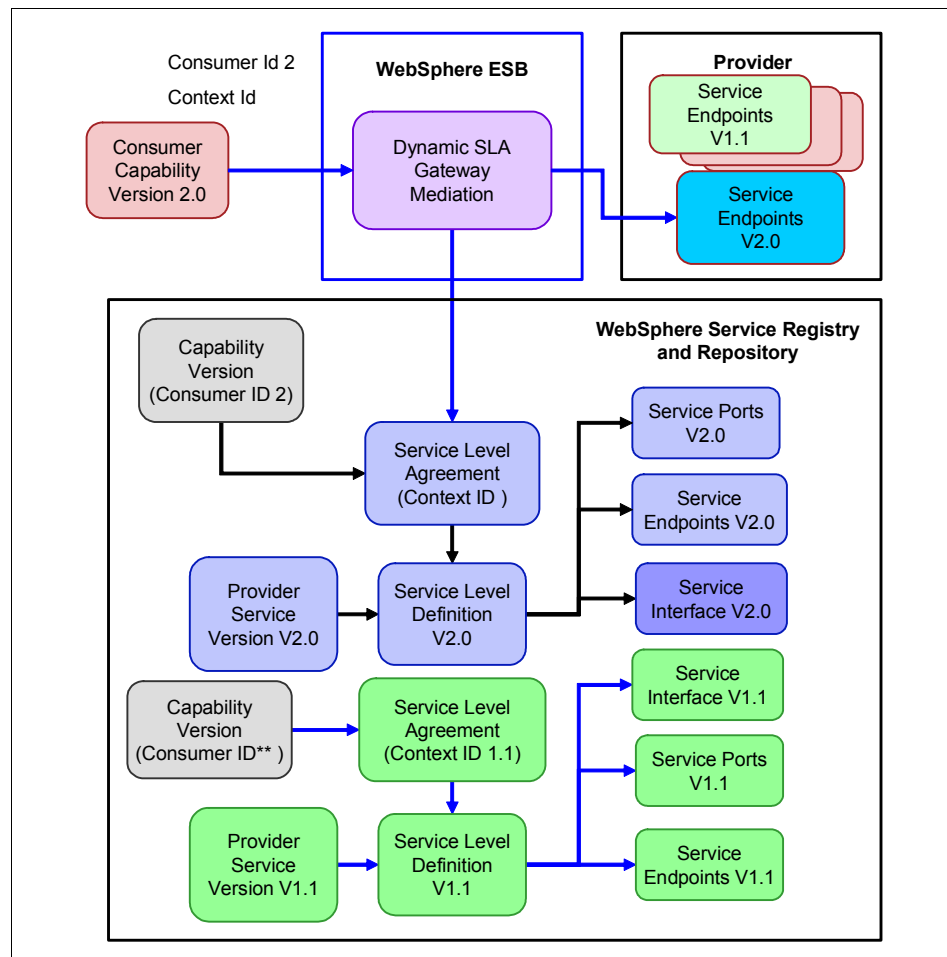


Figure 10-29 SLA-based look up to find a non-backward compatible service version 2.0

Configuring the v2.0 provider

To configure the service provider in WSRR, complete these tasks:

1. Follow the steps described in 7.4.3, “Registering a service” on page 220. Load the `AccountCreationInterfaceV2_0.wsdl` file as the interface WSDL for service version 2.0. Load the `AccountCreationV2_0_DevelopmentPort.wsdl` endpoint WSDL file.

2. Then, follow the steps described in 7.4.4, “Providing scope and planning information for a service” on page 229 and 7.4.5, “Providing a specification and service level definition for a service” on page 232.
3. At this point, the account creation service V2.0 is in the *Realized* state. Deploy the EAR file for the account creation service V2.0 (named `AccountCreationV2_0EAR.ear`) to a server in the development environment.
4. Classify the development service endpoint for V2.0 on test server with the environment classification *Development*.

Next, we configure the consumer application to consume the consumer service V2.0.

Configuring the consumer application v2.0

Follow the steps described in “Configuring the v1.1 consumer” on page 384 to create a consumer application version 2.0 called *Account Management Consumer* in WSRR. Assign a unique consumer ID of `AMC200` for this new version.

The consumer application version 2.0 is in the *Specified* state of the SOA life cycle environment. In addition, an SLA is in the *Active* state between this consumer application version 2.0 and the provider account creation v2.0. Specify the context ID property of the SLA to be `AMC200CI01`.

You are now ready to test by sending consumer requests to the account creation service v2.0 through the dynamic SLA gateway.

Testing

To test the SLA gateway solution for managing the service version 2.0 in the development environment, complete the following steps:

1. In IBM Integration Designer, open the context menu by right-clicking a WSDL in any web services project.
2. In the context menu, select **Web services** → **Test with Web Services Explorer**.
3. Clear the WSDL entry under WSDL main in the navigator view by clicking the eraser icon.
4. In the WSDL URL field, enter the following web services URL for the account creation to bring the WSDL into the navigator view:
`http://supply.devhost:9080/AccountCreationV2_0/services/AccountCreationServiceV2_0_DevelopmentPort?wsdl`
5. Click **Go**. Observe that ESB mediations route the new consumer request to the correct provider service version and end points per the SLA.

6. Select the endpoint, and click **Add**.
7. In the new second entry under Endpoints, copy and paste the service endpoint for the dynamic SLA gateway:
`http://supply.gatewayhost:9080/DynamicSLA_GatewayWeb/sca/DynamicSLA_GatewayExport_WS_SOAP11`
8. Select this entry, and click **Go** to set the endpoint to the gateway endpoint.
9. Next, open the source view for the input data for the createAccount operation. In the SOAP header window, enter the following XML text for the governance enablement profile gateway header:

```
<p:GEPGatewayHeader xmlns:p="http://com.ibm.wsrr.gep75/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.ibm.wsrr.gep75/schema
GEPGatewayHeader.xsd ">
  <consumerID>AMC200</consumerID>
  <contextID>AMC200CI01</contextID>
</p:GEPGatewayHeader>
```

10. Enter the following XML text for the body:

```
<q0:createAccount>
  <customer>
    <creditLimit>550</creditLimit>
    <active>true</active>
  </customer>
  <customer>
    <customerNumber>200</customerNumber>
    <firstName>Nay</firstName>
    <lastName>Boss</lastName>
  </customer>
  <repID>r123</repID>
</q0:createAccount>
```

Using the repID element: The addition of the repID element in the schema leads to the non-backward compatible change of the service interface.

11. Click **Go** to send the request for the account creation service to the dynamic SLA gateway.
12. Verify the response message as shown in Figure 10-30.

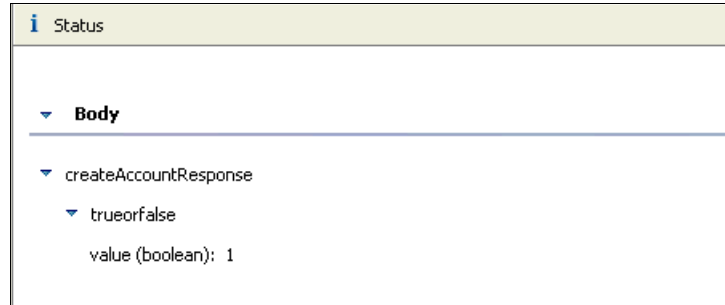


Figure 10-30 The response message from the account creation service V2.0 in the Web Services Explorer

13. Verify that the console view shows the output log of the WebSphere ESB Registry Edition server. The log displays the message as shown in Figure 10-31. This message indicates that the dynamic SLA gateway routes the request to the correct service version 2.0 endpoint.

```
[5/13/11 20:36:03:065 EDT] 000000cd SystemOut      O Service: Account Creation Service
[5/13/11 20:36:03:065 EDT] 000000cd SystemOut      O Version: 2.0
[5/13/11 20:36:03:065 EDT] 000000cd SystemOut      O Account for 300 created successfully!
```

Figure 10-31 The log message from account creation service version 2.0

10.4 Summary

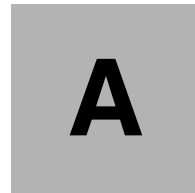
In SOA environments, multiple service versions can be running at the same time in a deployment environment. A service version in production can also exist in other non-production environments to facilitate development and testing.

The dynamic SLA gateway solution uses the SLA metadata model in the WSRR governance enablement profile and the WebSphere ESB SLA endpoint lookup primitive to manage service versions in such environments. WebSphere ESB Registry Edition enables smart SOA by managing service versions in diverse deployment scenarios, while providing governance of service artifacts throughout their life cycles.

Part 4



Appendixes



Additional material

This book refers to additional material that you can download from the Internet as described in this appendix.

Locating the material

The material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247949>

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials**, and open the directory that corresponds with the IBM Redbooks form number, SG247949.

Using the material

The additional material that accompanies this book includes the following folders:

<i>Folder name</i>	<i>Description</i>
Register_Service	Files supporting Chapter 7, “Registering services” on page 195
Basic_Mediation	Files supporting Chapter 8, “Implementing a mediation” on page 251
Extend_Mediation	Files supporting Chapter 9, “Extending the mediation” on page 315
Versioning	Files supporting Chapter 10, “Service versioning” on page 353

Downloading and extracting the material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material .zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 398. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
<http://www.redbooks.ibm.com/abstracts/sg246688.html?Open>
- ▶ *WebSphere Application Server V7: Concepts, Planning and Design*, SG24-7708
- ▶ *IBM Tivoli Composite Application Manager Family Installation, Configuration, and Basic Usage*, SG24-7151
- ▶ *Service Lifecycle Governance with IBM WebSphere Service Registry and Repository*, SG24-7793
- ▶ *Patterns for the Edge of Network*, TIPS0062
<http://www.redbooks.ibm.com/abstracts/tips0062.html?Open>

Online resources

For detailed information about system requirements, refer to the following links:

- ▶ WebSphere Enterprise Service Bus system requirements
<http://www.ibm.com/support/docview.wss?uid=swg27020614>
- ▶ WebSphere Service Registry and Repository system requirements
<http://www.ibm.com/software/integration/wsrr/sysreqs/#wsrrv75>
- ▶ IBM Integration Designer system requirements
<http://www.ibm.com/software/integration/integration-designer/sysreqs/>

You can find detailed information about this product and high availability at the following site:

- IBM WebSphere Application Server Information Center

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.edge.doc/lb/info/ae/welcome_overview.html

You can find more information about governance enablement profile models at the following site:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_models.html

You can find more information about the WSDL specification at the following site:

<http://www.w3.org/TR/wsd1>

You can find details about all the service model entities at the following site:

http://publib.boulder.ibm.com/infocenter/sr/v7r5/index.jsp?topic=/com.ibm.sr.doc/rwsr_gep_models.html

How to get IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers, Technotes, draft publications, and Additional materials, and order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Smart SOA Solutions with WebSphere Enterprise Service Bus Registry Edition V7.5

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Smart SOA Solutions with WebSphere Enterprise Service Bus Registry Edition V7.5

**Discover the
advantages of
integrating a service
registry with an ESB**

**Explore how to
register services and
implement
mediations**

**Learn by example
with practical
scenarios**

This IBM Redbooks publication provides you with a technical overview of WebSphere Enterprise Service Bus Registry Edition V7.5.

Part 1 outlines the roles of a service registry and an enterprise service bus (ESB), and explains the benefits of combining these technologies.

Part 2 focuses specifically on the ESB and registry that is offered by WebSphere Enterprise Service Bus Registry Edition. It also describes topology choices and installation.

Part 3 presents a fictional business scenario that demonstrates how an organization can register services and build simple and advanced mediations using these services.

IT specialists, IT architects, and those who are looking for a technical discussion of WebSphere Enterprise Service Bus Registry Edition will find value in this book.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks